Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Sistema d'obtenció i gestió remota en temps real de paràmetres mediambientals en un hivernacle

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Albert Cubiró Raventós

DIRECTORS: Rafael Vidal i Lluís Casals

DATA: 8 de setembre de 2017

Títol: Sistema d'obtenció i gestió remota en temps real de paràmetres mediambientals en un hivernacle

Autor: Albert Cubiró Raventós

Director: Rafael Vidal i Lluís Casals

Data: 8 de setembre de 2014

Resum

En els darrers anys, hi ha hagut un augment cada cop més significatiu de tecnologies (IEEE 802.15.4, 6LoWPAN, CoAP,etc.) per implementar xarxes de sensors (WSN, Wireless Sensors Network) on, petits dispositius compactes i econòmics, anomenats nodes-sensors, tenen la capacitat de mesurar, detectar i subministrar dades a altres sistemes.

Aquestes xarxes estan formades per nodes-sensors de baix cost i baix consum que es comuniquen entre sí de forma inalàmbrica, i estan especialment dissenyades per enllaços de comunicacions amb pèrdues i recursos limitats de memòria i de capacitat processat.

Aquest projecte neix de la col·laboració entre el Departament de Telemàtica de l'EETAC i el Departament d'Enginyeria Agroalimentària i Biotecnologia de l'ESAB, al Campus del Baix Llobregat de la UPC, per posar en marxa un sistema d'obtenció i gestió en temps real de paràmetres mediambientals en un hivernacle de l'Agròpolis situat a Viladecans (Barcelona).

Els paràmetres d'interès són la mesura de la temperatura i l'humitat a diferents profunditats del sòl, i la mesura de la temperatura ambiental en tres punts diferents de l'hivernacle. Per implementar aquest escenari, primer s'han dissenyat les interfícies per connectar els diferents sensors proporcionats (Decagon 5TE i 5TM, i RTD PT100) amb els nodes-sensors, implementats per la plataforma Econotag de Redwire. Després, s'ha creat una aplicació mitjançant el SO Contiki, basada en el procotol CoAP (Constrained Application Protocol), per llegir i gestionar les dades dels sensors de cada node-sensor. Finalment, s'ha creat una xarxa de sensors basada en l'estàndard IEEE 802.15.4 per recuperar les dades dels diferents sensors i pujar-les a Internet.

El router encarregat de donar connectivitat IPv6 als nodes de la WSN està format per una Raspberry Pi B+ (RPi) més un mòdul per proporcionar-li connectivitat IEEE 802.15.4. La RPi realitza peticions periòdiques als diferents nodes-sensors per obtenir els paràmetres mesurats, emmagatzemar-los a la memòria local i pujar-los a un full de càlcul de la plataforma Google Drive.

Title: Sistema d'obtenció i gestió remota en temps real de paràmetres mediambientals en un hivernacle

Author: Albert Cubiró Raventós

Director: Rafael Vidal i Lluís Casals

Date: September 8 th 2017

Overview

In recent years, there has been a significant increase in some technologies (IEEE 802.15.4, 6LoWPAN, CoAP, etc.) to implement wireless sensor networks (WSN). Where, small, compact and economic devices, called nodes-sensors, have the ability to measure, detect and supply data to other systems.

These networks are formed by low cost and energy-saving nodes-sensors that communicate wirelessly. They are specially designed for communication with lossy links, and limited memory resources and processing capacity.

This project is based on the collaboration between the Department of Telematics of EETAC and the Department of Agri-Food Engineering and Biotechnology of ESAB, at the UPC Campus in Baix Llobregat. The purpose is to deploy a system to collect and manage, in real-time, environmental parameters in an Agropolis greenhouse located in Viladecans (Barcelona).

The parameters of interest are the measurement of temperature and humidity at different soil depths, and the measurement of the ambient temperature in three different points of the greenhouse. To implement this scenario, the interfaces were first designed to connect the different sensors provided (Decagon 5TE and 5TM, and RTD PT100) with the nodes-sensors, implemented by the Redwire Econotag platform. Then, an application has been created using the Contiki OS, based on the CoAP protocol (Constrained Application Protocol), to read and manage sensor data on each sensor node. Finally, a WSN based on the IEEE 802.15.4 standard has been created to recover the data from the different nodes-sensors and upload them to the Internet.

The router in charge of giving IPv6 connectivity to the WSN nodes is formed by a Raspberry Pi B + (RPi) plus a module to provide IEEE 802.15.4 connectivity. The RPi performs periodic requests to the different nodes-sensors to obtain the measured parameters, store them in the local memory, and upload them to a spreadsheet on the Google Drive platform.

ÍNDEX

ÍNDEX DE FIGURES:

ÍNDEX DE TAULES:

INTRODUCCIÓ

Aquest projecte neix de la col·laboració entre el Departament de Telemàtica de l'EETAC i el Departament d'Enginyeria Agroalimentària i Biotecnologia de l'ESAB, al Campus del Baix Llobregat de la UPC, per posar en marxa un sistema d'obtenció i gestió en temps real de paràmetres mediambientals en un hivernacle de l'Agròpolis[1] a Viladecans (Barcelona).

Els paràmetres principals d'interès són la temperatura i la humitat del sòl. Per mesurar-los es disposa dels sensors Decagon 5TE i 5TM. També es disposa dels sensors analògics PT-100, emprats per l'obtenció de la temperatura ambiental.

Aquest projecte té com a primer objectiu el disseny i realització de les interfícies hardware (HW) i software (SW) per poder interactuar amb els sensors mencionats.

El següent objectiu és crear una xarxa de sensors (en anglès Wireless Sensor Network, WSN) que permeti recuperar les dades obtingudes pels diferents sensors i accedir a elles mitjançant el protocol IP (Internet Protocol). Concretament, IPv6 per la versió de referència per la seva utilització en les WSN i, en general, a l'Internet de les Coses (Internet of Things, IoT).

L'últim objectiu és automatitzar el procés d'obtenció dels paràmetres mediambientals i la pujada de les dades obtingudes a un full de càlcul de la plataforma Google Docs. Per assolir aquest darrer objectiu és clau el fet que a l'hivernacle es disposi d'accés a Internet via Wi-Fi.

Aquesta memòria comença amb un primer capítol introductori que presenta el problema a solucionar amb més detall i descriu els diferents elements que formen el sistema que es vol implementar. Segueix amb els capítols dos i tres que expliquen el disseny de les interfícies per connectar els sensors Decagon 5TE i 5TM, i els sensors PT-100.

Un cop explicades les interfícies, el quart capítol explica el procés de posada en marxa de les diferents parts de la WSN. El cinquè capítol mostra com s'automatitzen els processos d'obtenció dels paràmetres mediambientals requerits, i de pujada d'aquestes dades a Internet. Per últim, hi ha un sisè capítol de conclusions i línies futures, i diferents annexos.

[1] http://www.upc.edu/parcupc/espais/viladecans/agropolis/.

CAPÍTOL 1. PRESENTACIÓ DEL PROBLEMA I REQUERIMENTS

En aquest capítol inicial es descriuran els requeriments i objectius a complir al llarg d'aquest projecte, així com les diferents parts que formen el sistema.

1.1. Escenari

Els requeriments d'aquest sistema són obtenir els paràmetres de temperatura i humitat del sòl a diferents profunditats, i la temperatura ambiental en tres punts diferents de l'hivernacle. L'obtenció d'aquests paràmetres s'ha de fer de forma automàtica i, un cop s'han obtingut, s'ha de transferir aquesta informació a un full de càlcul de Google Drive per guardar-la i tractar-la posteriorment, per l'usuari final.

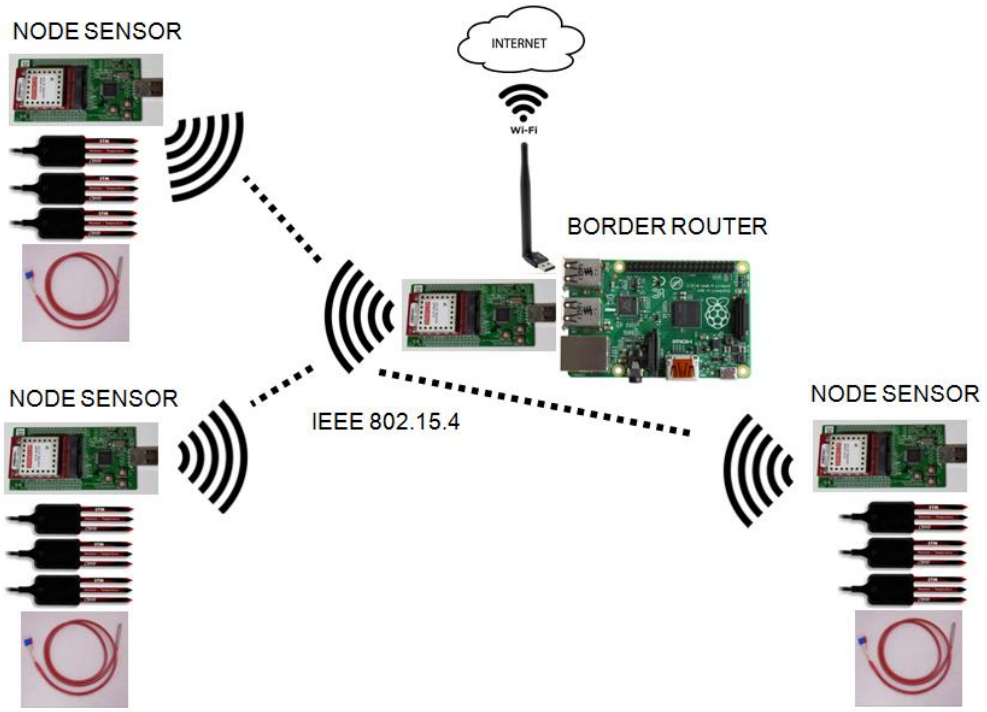

Fig. 1.1 Escenari. Xarxa de sensors a desplegar a l'hivernacle de l'Agròpolis.

A la figura 1.1 es veuen els diferents elements que formen la xarxa de sensors que es vol instal·lar a l'hivernacle de l'Agròpolis. A continuació s'indiquen les característiques més importants de cada element.

- **Node sensor.** Disposa d'un mòdul ràdio IEEE 802.15.4 i s'alimenta amb bateries. Suporta els protocols 6LoWPAN[2], IPv6, RPL[3] i CoAP[4]. Cada node sensor incorpora 3 sensors digitals de la marca Decagon (models 5TE i 5TM) per mesurar, en el sòl, les magnituds físiques següents: Electroconductivitat (EC), Volum de contingut d'aigua (VWC) i Temperatura. Cada node sensor també incorpora un sensor analògic PT100 per mesurar la temperatura ambiental a l'interior de l'hivernacle.

- **Border router.** Disposa d'una interfície IEEE 802.15.4 i s'alimenta amb presa de corrent disponible a l'hivernacle. Suporta els protocols: 6LoWPAN, IPv6, RPL i CoAP, entre altres. Disposa d'una interfície IEEE 802.11 per connectar-se a la xarxa Wi-Fi disponible a l'hivernacle i així, poder-se connectar a Internet.

En els següents apartats d'aquest capítol es descriurà el hardware i el software escollit per implementar aquests elements.

Per extreure la informació dels diferents nodes sensors de la WSN i poder-la accedir des de qualsevol punt amb accés a Internet s'utilitzarà una arquitectura client - servidor amb diversos protocols dedicats a dispositius de baix cost, curt abast i de baix consum.

A les figures 1.2 i 1.3 es poden veure les dues piles de protocols presents en el sistema a desenvolupar.

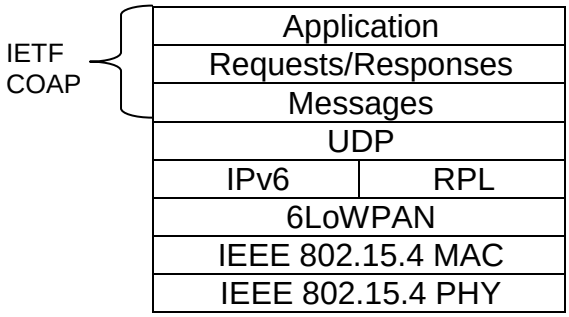

Fig. 1.2 Pila de protocols 6LoWPAN.

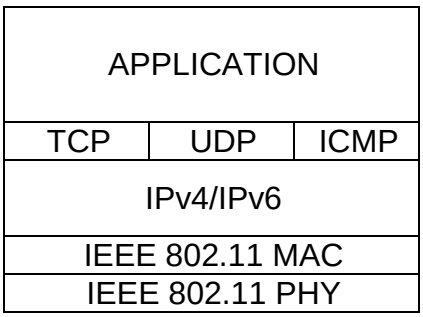

Fig. 1.3 Pila de protocols TCP/IP.

[2] **6LoWPAN.** És un protocol de comunicacions sense cables via ràdio-freqüència que segueix la recomanació RFC 4944 de l'IETF. Està destinat a crear xarxes d'àrea personal de baixa potència i baixa velocitat de transmissió de dades.

[3] **RPL**. Protocol estanderditzat pel grup de treball ROLL (Routing over Low-Power and Lossy Networks) de l'IETF. Alguns dels objectius del protocol RPL són minimitzar la despesa d'energia, minimitzar la latència o satisfer requeriments determinats (RFC 6550).

[4] **CoAP.** És un protocol de la capa d'aplicació d'Internet per a dispositius amb recursos restringits que segueix la recomanació RFC 7252 de l'IETF. Permet que els dispositius amb pocs recursos es puguin comunicar amb qualsevol node d'Internet.

Com es pot veure a les figures 1.2 i 1.3, a nivell físic i a nivell MAC s'utilitza el protocol IEEE 802.15.4. La freqüència de treball serà a la banda ISM sense llicència de 2.4 GHz[5], ja que el mòdul ràdio que incorporen les Econotags treballa amb aquest protocol i només a aquesta banda.

A nivell d'adaptació de la xarxa, el protocol utilitzat és el 6LoWPAN. Aquest protocol és necessari per a poder enviar paquets IPv6 sobre un enllaç IEEE 802.15.4. Com a protocol d'encaminament s'utilitza l'RPL, que permet diverses topologies, així com diferents tipus de tràfic (punt a multipunt, multipunt a punt i punt a punt).

A nivell d'aplicació s'empra el protocol CoAP, que és un estàndard obert de l'IETF[6], dissenyat per a dispositius amb poca memòria i capacitat de processat. També està orientat a aplicacions M2M[7] tals com projectes de gestió intel·ligent de l'energia i automatització d'edificis.

Per poder extreure les dades de la WSN i visualitzar-les des d'Internet IPv6 necessitem algun sistema que faci la conversió d'una pila de protocols a l'altra. Aquest sistema és el border router (BR).

A continuació es farà una breu descripció dels diferents elements HW i SW presents en aquest sistema. En capítols posteriors es veurà amb més profunditat cada element, el seu funcionament i com es configura.

1.2. Hardware

En aquest apartat es veuran les característiques i prestacions dels diferents elements HW que intervenen en el sistema desplegat a l'hivernacle. Primer, es veuran les característiques dels sensors encarregats de llegir els paràmetres de temperatura i humitat requerits, després es veuran les característiques dels nodes encarregats de llegir les dades dels sensors i les característiques del router encarregat de comunicar la WSN amb Internet.

Segons els requeriments citats a l'apartat 1.1, els sensors utilitzats per mesurar, en el sòl, les magnituds EC, VWC i Temp, són els models 5TE i 5TM de la marca Decagon. El sensor analògic PT100 s'utilitza per mesurar la temperatura ambiental de l'hivernacle.

[5] L'estàndard IEEE 802.15.4 contempla poder operar en tres bandes ISM. 868 MHz (Europa), 900 (Amèrica del Nord) i 2450 MHz (A tot el món).

[6] L'**Internet Engineering Task Force** (**IETF**) és una organització que desenvolupa i promou estàndards, tractant en particular els estàndards del protocol TCP/IP i la suite de protocols d'Internet.

[7] **M2M.** Acrònim anglès de Màquina a Màquina. Es refereix a comunicacions directes entre dues màquines i en qualsevol tipus de canal (amb cables o sense fils).

1.2.1. Sensors Decagon

A la figura 1.4 es pot veure com és el sensor 5TM físicament.

Fig. 1.4 Sensor de temperatura i VWC marca Decagon i model 5TM.

Les característiques més destacables del sensor Decagon 5TM són les següents:

- Tensió de treball: 3.6 Vcc – 15 Vcc.
- Corrent màxima: 10 mA durant els 150 ms que dura la mesura.
- Sortida: Digital RS-232 (TTL).
- Temperatura d'operació: -40 ºC – 60 ºC.
- Resolució: 0.1 ºC.
- Exactitud: ± 1 ºC.
- Rang del VWC: 1 – 80 (es mesura a partir de la permitivitat dielèctrica aparent del medi ε_a, l'aire equival a 1 i l'aigua a 80).
- Resolució del VWC: ε_a: 0.1 entre 1 – 20 ε_a, < 0.75 entre 20 – 80 ε_a. VWC: 0.08%.
- Exactitud del VWC: ± 1 ε_a entre 1 – 40, ± 15% VWC entre 40 i 80.

Les característiques del sensor 5TE són les mateixes que el sensor 5TM afegint la magnitud EC que el sensor 5TM no mesura. A continuació veiem les prestacions en la mesura de l'EC:

- Rang: 0 – 23 dS/m.
- Resolució: 0.01 dS/m entre 0 i 7 dS/m, 0.05 dS/m entre 7 i 23 dS/m.
- Exactitud: ± 10 % entre 0 i 7 dS/m, es requereix calibratge de l'usuari entre 7 i 23 dS/m.

1.2.2. Sensor RTD PT100

El sensor PT100 proporcionat és el de la figura 1.5.

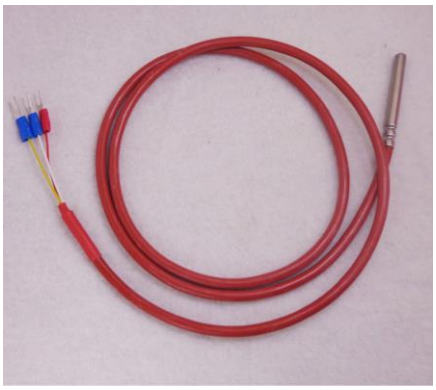

Fig. 1.5 Sensor RTD PT100.

Les característiques més importants del sensor PT100 són les següents:

- Tensió de treball: 5 Vcc.
- Corrent màxima: 1 mA (mínima per prevenir l'autoescalfament).
- Sortida: Analògica. Resistència proporcional a la temperatura mesurada.
- Temperatura d'operació: -50 ºC – 230 ºC.
- Resolució: < 0.6 ºC entre -10 ºC – 45 ºC.
- Exactitud: ± 1 ºC.

1.2.3. Node Econotag

Els nodes encarregats de llegir, processar i transmetre les dades dels sensors seran de la marca Redwire model Econotag M12. No obstant, hem de diferenciar dues coses, la primera, és què aquesta placa consta d'una unitat de control principal que és la pròpia placa M12 (que inclou el xip mc13224v, del fabricant Freescale, amb el mòdul ràdio ja incorporat). La segona, és què la placa Econotag és la versió II i la revisió C. Això es pot observar a la figura 1.6.

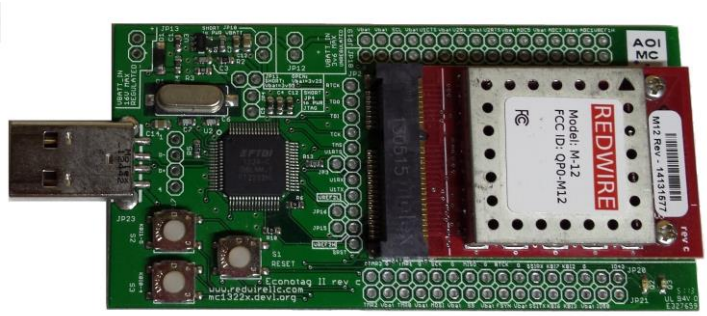

Fig. 1.6 Plataforma Econotag II rev. C de la marca REDWIRE.

Si s'observa la figura 1.6 es pot veure que la placa vermella és la M12 (inscripció REDWIRE), mentre que la placa verda és l'Econotag pròpiament. La plataforma Econotag permet accedir més fàcilment als ports d'entrada/sortida (GPIO, General Purpose Input/Output) del xip mc13224v, afegeix funcionalitats addicionals com el conversor de port sèrie a USB, un port JTAG[8] per depuració i un convertidor DC-DC de tipus Buck.

Les característiques més importants del xip mc13224v són les següents:

- Unitat de Control Principal (les seves sigles en anglès són MCU): ARMv7 32 bits.
- Freqüència màxima del rellotge: 26 MHz.
- Memòria: 96 Kb de RAM i 128 Kb de memòria FLASH.
- Voltatge d'operació: De 2.0 Vcc a 3.6 Vcc.
- Consum en repòs (sense el mòdul ràdio engegat): 0.8 mA.
- Consum amb el MCU actiu i el mòdul ràdio apagat: 3.3 mA.
- Temperatura d'operació: De -40 ºC a 105 ºC.
- Transceptor/mòdem que compleix l'estàndard IEEE 802.15.4:
  - Banda d'operació ISM 2.4 GHz.
  - 16 canals disponibles.
  - Potència de sortida programable (de -30 dBm a +4 dBm).
  - Sensibilitat de fins a -100 dBm[9].
  - 22 mA de consum típic de corrent en RX.
  - 29 mA de consum típic de corrent en TX.
- Dos mòduls ADC de 12 bits amb 8 canals d'entrada.
- Fins a 64 GPIO's d'entrada/sortida configurables.
- Dos mòduls UART.

[8] **Port JTAG.** Join Test Action Group (JTAG) és un grup format per fabricants d'electrònica d'arreu que van decidir trobar una solució comuna al problema de depuració de programari. La solució va esdevenir en l'estàndard 1149.1-1990 de l'IEEE Standard Test Access Port and Boundary-Scan Architecture. L'IEEE Std 1149.1 permet que instruccions de test i dades puguin ser carregades a la memòria dels dispositius i posteriorment recollir els resultats dels tests de la memòria del dispositiu.

[9] Molt per sobre del què marca l'estàndard IEEE 802.15.4, que és de -85 dBm.

1.2.4. Raspberry Pi B+

S'ha escollit la Raspberry Pi B+ (RPi) per ser una opció econòmica i potent (veure referència bibliogràfica [1]), alhora que és un dispositiu àmpliament utilitzat com a base per a múltiples solucions de xarxa com ara el software 6LBR que es descriu a l'apartat 1.3.

La RPi disposa de dues interfícies ràdio, una pel protocol IEEE 802.11 (Wi-Fi) que li permet obtenir accés a Internet mitjançant un router Wi-Fi disponible a l'hivernacle, i l'altra, pel protocol IEEE 802.15.4 per crear la WSN i comunicar-se amb els diferents nodes. A continuació es descriu el HW utilitzat per disposar d'aquestes interfícies, ja que la RPi no les incorpora de fàbrica.

1.2.4.1. Interfície IEEE 802.15.4

L'Econotag amb el programa *slip-radio.bin*[10] carregat a la memòria flash s'utilitza com a interfície IEEE 802.15.4 per a la RPi. Quan es connecta l'Econotag a un dels ports USB de la RPi, i es configura la RPi adequadament, aquest programa permet capturar els paquets de la WSN i enviar-los pel port sèrie a la RPi per al seu processat. A l'ANNEX IX d'aquesta memòria es pot revisar la configuració d'ambdues parts.

1.2.4.2. Interfície IEEE 802.11

Com a targeta Wi-Fi s'ha utilitzat el model BL-WN155A del fabricant LB-LINK. A continuació es poden veure les característiques més destacables:

- Adaptador USB compatible amb els protocols IEEE 802.11b/g/n.
- Velocitat màxima de transmissió de dades de 150 Mbps.
- Antena incorporada amb un guany màxim de 5 dBi.
- Potència màxima de transmissió de 17 dBm (PIRE[11] màxima).
- Suporta encriptació WEP, WPA/WPA2 i WPA-PSK/WPA2-PSK (TKIP/AES) de 64 i 128 bits.

1.3. Software

Degut a les diferents arquitectures presents al sistema, aquest apartat s'ha dividit, per una banda, amb el SW involucrat amb els nodes Econotag, i per l'altra, amb el SW involucrat amb la RPi.

[10] Aquest programa captura paquets per l'interfície IEEE 802.15.4 i els envia pel port sèrie mitjançant protocol IP (Serial Line IP, SLIP).
[11] **PIRE.** Potència Isotròpica Radiada Equivalent.

1.3.1. Node Econotag

Degut a les limitacions de memòria i de capacitat de processat dels nodes Econotag, el software elegit ha sigut el sistema operatiu (SO) Contiki versió 2.7. Aquest és un SO de distribució lliure i dissenyat per a l'Internet de les Coses (Internet Of Things, IoT). Aquest SO implementa els protocols de l'IETF per a l'IoT que volem fer servir en el nostre escenari. Això és, IPv6, la capa d'adaptació 6LoWPAN, el protocol RPL IPv6 d'encaminament i el protocol d'aplicació CoAP RESTful. A l'annex VIII d'aquesta memòria es pot revisar l'estructura de carpetes del SO.

Per programar els nodes Econotag s'utilitza un entorn ja preparat i configurat que es pot descarregar des de l'enllaç[12]. A l'annex I d'aquesta memòria es pot revisar la guia per programar els nodes Econotag, així com diferents consells a l'hora de treballar amb aquesta plataforma.

1.3.2. RaspberryPi B+

Per utilitzar la RPi com a border router (BR) s'ha utilitzat el SO Raspbian[13], concretament la versió Jessie amb data 25 de novembre de 2016. A més, s'han instal·lat un conjunt de paquets i llibreries que es mencionen a continuació.

El SW 6LBR és una solució per a BR basada en 6LoWPAN/RPL. Està dissenyat per ser flexible i suportar diverses topologies de xarxa. A més, és un software lliure. Al capítol 4 d'aquesta memòria es veu més detalladament el funcionament d'aquest SW en conjunt amb el HW incorporat a la RPi.

Per executar el codi per realitzar les peticions CoAP des del propi BR i pujar les dades obtingudes a Internet són necessaris els següents paquets i llibreries:

- Paquet Python v2.7.13.
- Twisted Framework. Infraestructura de xarxa, per a programació dirigida per esdeveniments, escrita en llenguatge *Python* i sota la llicència MIT. *Twisted* proporciona suport per diverses arquitectures (TCP, UDP, SSL/TLS, IP Multicast, etc.).
- Llibreria Txthings. És una llibreria CoAP escrita en llenguatge *Python* per al framework *Twisted*.
- Llibreria GData Python Client. Per escriure una aplicació utilitzant un dels serveis de dades de Google.
- Llibreria GSpread. Per operar dins dels fulls de càlcul de Google Drive.
- Llibreria oauth2client. Per autenticar-te a la plataforma Google Drive.

[12] https://sourceforge.net/projects/contiki/files/latest/download?source=files
[13] **SO Raspbian.** Basat en la distribució de Linux Debian, i que s'ha optimitzat pel hardware de la RaspberryPi.

CAPÍTOL 2. INTERFÍCIE ECONOTAG – SENSORS DECAGON

Un cop presentats els elements que intervenen en l'escenari, passem a desgranar el disseny de les interfícies per connectar els sensors digitals 5TE i 5TM de Decagon. Primer, s'explica com es connecten físicament els sensors amb els nodes Econotag, després, es veu com s'ha solucionat el control de l'alimentació dels sensors, i la compartició del bus de dades.

Després de veure la part HW es veurà el SW que realitza les lectures dels sensors i que gestiona les dades rebudes.

2.1. Intercomunicació Econotag - sensors

A la figura 2.1 es pot veure l'esquema d'interconnexió del sensors 5TE i 5TM de la marca Decagon.

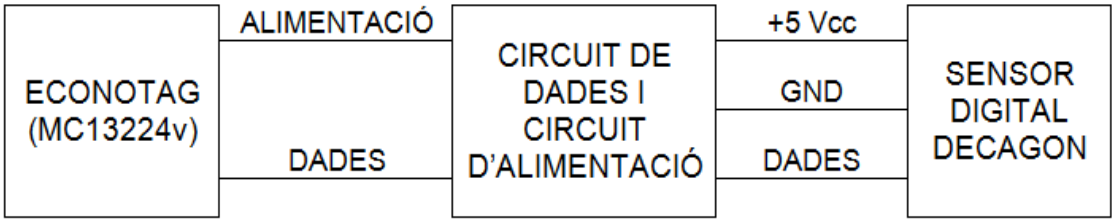

Fig. 2.1 Esquema de connexió dels sensors digitals 5TE i 5TM.

Com es pot observar a l'esquema de la figura 2.1, els sensors tenen 3 terminals de connexió (+Vcc, GND i Dades). El driver físic del sensors no es resumeix a un circuit de condicionament tradicional, sinó que hi ha dos circuits diferenciats, un pel control de l'alimentació, i l'altre, per la transmissió de les dades.

Un cop revisada la documentació del sensors (veure referències bibliogràfiques [2] i [3]) veiem que envien caràcters a una velocitat de 1200 bps. Cada caràcter té un bit d'inici, 8 bits de dades (LSB primer), sense paritat i un bit d'aturada. El bus és actiu alt o de nivells lògics no invertits.

A la figura 2.2 es pot veure un senyal d'exemple de les dades que ens proporcionen els sensors.

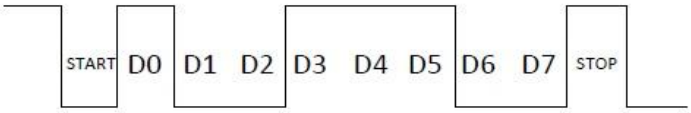

Fig. 2.2 Exemple de transmissió sèrie del caràcter '9' (0x39).

Per obtenir les dades dels sensors s'ha de tenir en compte que aquests sensors permeten treballar amb dos protocols de comunicació: Sèrie[14] o SDI-12[15]. La seqüència que segueix el senyal en el domini temporal es pot veure a la figura 2.3.

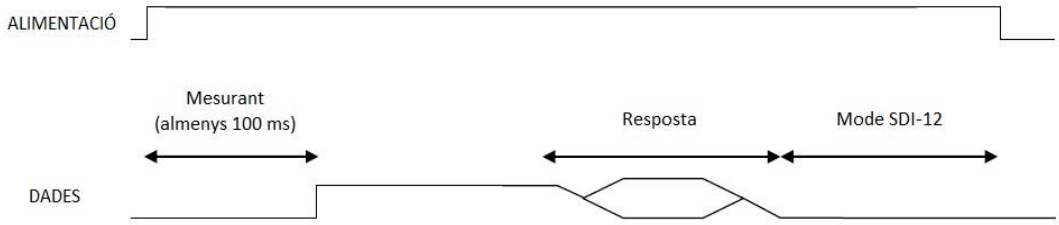

Fig. 2.3 Seqüència del senyal que proporcionen els sensors.

Quan s'alimenta el sensor, aquest manté el bus de dades en estat baix mentre efectua la mesura. Un cop ha finalitzat, porta el bus de dades a l'estat alt per indicar que ha acabat la mesura i que procedirà a enviar les dades. Primer les envia mitjançant el protocol sèrie i, un cop ha finalitzat l'enviament, canvia al protocol SDI-12. Per tornar a rebre dades d'una altra lectura mitjançant el protocol sèrie necessitem treure l'alimentació al sensor i alimentar-lo de nou.

El format de les dades que es reben del sensor (caràcters) hauria de tenir un format com el de la figura 2.4.

```
56 432 645<0D>zJ<0D><0A>
```

Fig. 2.4 Format de les dades obtingudes del sensor Decagon 5TE.

[14] **Protocol sèrie.** Dissenyat per l'intercanvi de dades entre un Equip Terminal de Dades i un Equip de Comunicació de Dades. Per exemple entre un PC i un perifèric. Utilitza un protocol de transmissió asíncron i la tecnologia TTL (en anglès, Transistor – Transistor Logic) que defineix els següents nivells lògics: Estat Baix (comprès entre 0 Vcc i 0.8 Vcc) i Estat Alt (comprès entre els 2.2 Vcc i Vcc).

[15] **Protocol SDI-12 (en anglès, Serial Digital Interface at 1200 baud).** Dissenyat per l'intercanvi de dades entre sensors intel·ligents i equips enregistradors de dades. És un protocol de comunicació sèrie asíncron que segueix una configuració màster – esclau on, un enregistrador de dades demana les dades dels sensors (fins a 62) identificats amb una adreça única.

A la taula 2.1 es pot veure una descripció de cada camp de la trama de dades enviada pel sensor 5TE.

Taula 2.1 Descripció dels paràmetres enviats pels sensors 5TE i 5TM.

PARÀMETRE	DESCRIPCIÓ
56	Constant dielèctrica per processar amb el format $r_\varepsilon = \varepsilon \cdot 50$. Per convertir-la a VWC en sòl mineral, el fabricant recomana utilitzar l'equació Topp: $\Theta = (4.3 \cdot 10^{-6} \cdot \varepsilon^3) - (5.5 \cdot 10^{-4} \cdot \varepsilon^2) + (2.92 \cdot 10^{-2} \cdot \varepsilon) - (5.3 \cdot 10^{-2})$ En l'exemple, 56 / 50 = 1.12 Aproximadament, el valor de la constant dielèctrica de l'aire ($\varepsilon = 1$).
432	Conductivitat elèctrica (en anglès, EC) en *dS/m* multiplicat per 100. Dividint aquest número per 100 aconseguim la EC en *dS/m* (o *mS/cm*). Aquest valor té corregida la temperatura mitjançant el propi sensor de temperatura del sensor 5TE. Valors de EC per processar en aigua de l'aixeta poden anar de 10 a 80 (0.1 *dS/m* a 0.8 *dS/m*). En l'exemple, 432 és la EC per processar llegida. Dividint aquest valor per 100 ens dóna un resultat de 4.32 *dS/m*. Per valors per processar que superin el valor de 700, primer necessitem realitzar la següent descompressió: $EC_{decompressed} = 5 * (EC_{raw} – 700) + 700$
645	Temperatura per processar (T_{raw}). $T_{raw} = (10 \cdot T) + 400$. On T és el valor de temperatura en graus Celsius. En l'exemple, 645 - 400 = 245; 245 / 10 = 24.5 °C. Per valors de T_{raw} que excedeixin de 900 hem de fer aquesta conversió abans de fer la conversió a graus Celsius: $T_{raw}^{I} = 5 \cdot (T_{raw} - 900) + 900$

<0D>	Aquest caràcter de retorn de carro marca el final de l'string de les mesures i l'inici de l'string de metadades.

Els següents camps són metadades:

z	Caràcter per indicar el model del sensor. La 'z' indica que és el sensor 5TE.
J	Caràcter de checksum. El fabricant l'utilitza en els seus instruments per assegurar que les dades transmeses són vàlides. El checksum es calcula sobre el conjunt de caràcters següent: 56 432 645<0D>z
<0D><0A>	Els caràcters de retorn de carro i de salt de línia marquen el final de les metadades i la fi de la transmissió.

La resposta del sensor 5TM és la mateixa que el sensor 5TE, com el 5TM no mesura la EC, aquest valor sempre serà igual a 0.

2.2. Control de l'alimentació

Per poder realitzar lectures successives dels sensors s'ha de controlar l'alimentació d'aquests. Per aquest objectiu s'ha escollit un disseny amb dos transistors treballant en la zona de saturació. Després de valorar les diferents opcions de disseny possibles s'ha escollit el tall del pol positiu d'alimentació com a millor opció, ja que l'altra opció valorada (tall del pol negatiu d'alimentació) provoca que els sensors estiguin permanentment alimentats i tinguin la massa flotant. Això pot provocar que els sensors enviïn caràcters erronis i el receptor UART de l'Econotag no els interpreti com a caràcters imprimibles.

S'ha descartat alimentar els sensors directament des dels pins del GPIO perquè, després de mesurar el consum de corrent d'un sensor 5TM mitjançant el circuit de la figura 2.5, s'ha comprovat que en l'inicialització d'aquest, es produeix un pic de corrent superior als 10 mA, valor màxim d'intensitat de sortida del xip *mc13224v* (MCU de l'Econotag).

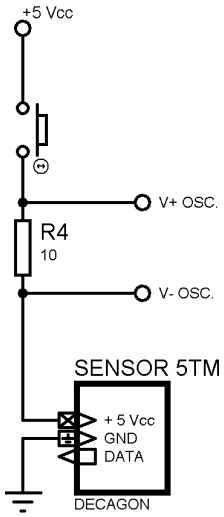

Fig. 2.5 Circuit per mesurar el consum de corrent del sensor 5TM.

Després de revisar la documentació del sensor i de la MCU de l'Econotag (veure referència bibliogràfica [4]), els requeriments inicials del sistema són els següents:

- V_{out} màxima de l'Econotag (mc13224v): 80% V_{BATT} < V_{BATT} (~ 3.3 V_{cc}).
- I_{out} màxima de l'Econotag (mc13224v): 10 mA.
- Mínim consum i baixa resistència quan el transistor condueix R_{DS} (ON) ≅ 0 Ω.
- V_{in} del sensor 5TM: 3.6 V_{cc} < V_{in} < 15 V_{cc}.
- $I_{màx}$ del sensor 5TM: 10 mA.

Per tant, el transistor que sigui operat directament pel pin del GPIO de l'Econotag ha de poder treballar dintre dels marges de tensió i corrent anteriorment citats, mentre que el transistor que operi l'alimentació del sensor ha de ser capaç d'operar dins dels marges de tensió i corrent del sensor.

El disseny del circuit per controlar l'alimentació del sensor 5TM és el de la figura 2.6.

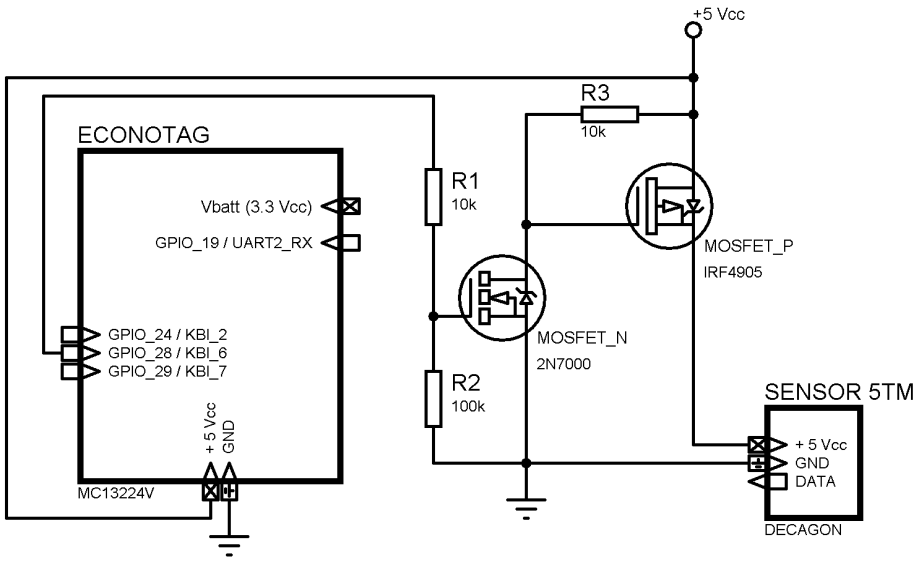

Fig. 2.6 Circuit de control de l'alimentació d'un sensor Decagon 5TM.

Per escollir els transistors del circuit de la figura 2.6 s'han seguit els requeriments anteriorment citats. Així, s'ha escollit com a transistor *MOSFET*[16] de canal N, el model 2N7000 de Fairchild Semiconductor (veure referència bibliogràfica [5]), i com a transistor *MOSFET* de canal P, el model IRF4905 de International Rectifier (veure referència bibliogràfica [6]).

Les resistències R1 i R3 són per limitar el corrent de porta (en anglès Gate - G) a cada transistor, mentre que la resistència R2 és per definir l'estat baix (0 V_{cc}) correctament quan el MCU s'inicialitza.

A l'annex VII d'aquesta memòria es pot revisar el mètode utilitzat per comprovar que el circuit de la figura 2.6 funciona adequadament.

2.3. Bus de dades compartit

El disseny final del sistema inclou 3 sensors digitals Decagon connectats a cada Econotag, i aquesta només disposa de dos busos UART. Per aquest motiu s'ha hagut de dissenyar un circuit que permeti compartir un sol bus UART pels tres sensors que han d'enviar les dades a través seu, ja que l'altre bus UART s'utilitza per depurar el programa en temps d'execució.

A l'annex III d'aquesta memòria es pot revisar el procediment per operar els mòduls UART de l'Econotag.

[16] ***MOSFET*** (Metal Oxide Semiconductor Field Effect Transistor). O transistor d'efecte de camp mitjançant semiconductor òxid.

Després de revisar el datasheet del xip *mc13224v* i la guia d'integració dels sensors 5TE i 5TM, el circuit proposat està format per 3 díodes model 1N4007 (veure referència bibliogràfica [7]) i 4 resistències i es pot revisar a la figura 2.8.

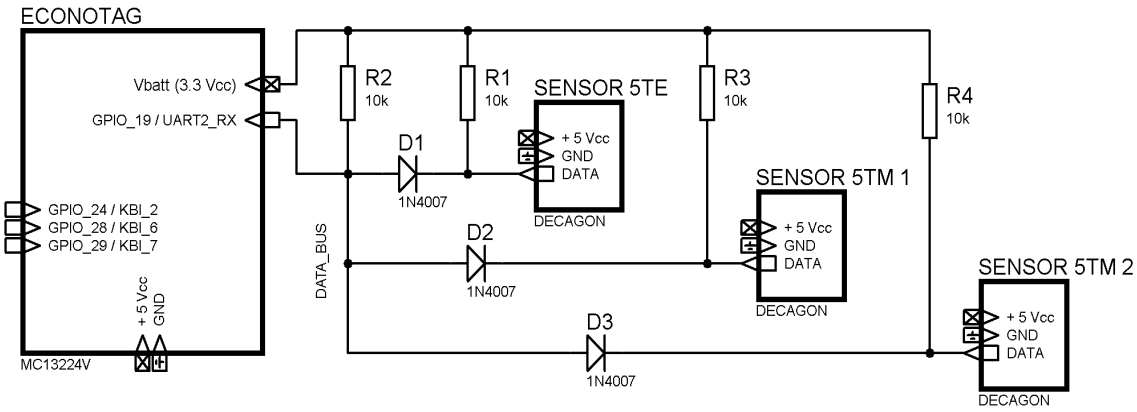

Fig. 2.7 Circuit per compartir el bus de dades.

El circuit de la figura 2.8 posa a nivell alt el bus de dades mitjançant les resistències de 10 kΩ connectades al pol positiu de l'alimentació de l'Econotag (~ +3.3 Vcc). Això és degut a què, com s'ha dit, el bus treballa amb nivells lògics no invertits o estat alt. Amb els díodes aconseguim aïllar el circuit de dades de cada sensor. Un cop fet això, si alimentem els sensors per separat, d'un en un, podem obtenir les dades de cada sensor de forma individual, ja que quan el sensor enviï les dades, cada vegada que posi el bus a nivell 0, el díode condueix i l'Econotag ho interpreta com un 0.

Els pins dels GPIO detallats a l'esquerra de la figura 2.8, corresponen als pins dedicats al control de l'alimentació dels sensors. El GPIO dedicat al control de l'alimentació del sensor 5TE és el GPIO_24/KBI_2. El GPIO_28/KBI_6 està dedicat al sensor 5TM 1 i, el GPIO_29/KBI_7 està dedicat al sensor 5TM 2.

A l'annex VI d'aquesta memòria es pot revisar la placa prototipus que s'ha muntat, amb els components dels circuits esmentats soldats.

2.4. Lectura i processat de les dades

El diagrama de blocs definit per al maneig, l'adquisició i el processament de les dades dels sensors és el de la figura 2.9. Cal destacar que aquest diagrama fa referència al maneig d'un sensor, ja que la lectura dels sensors 5TE i 5TM es realitza de forma seqüencial, i d'un en un.

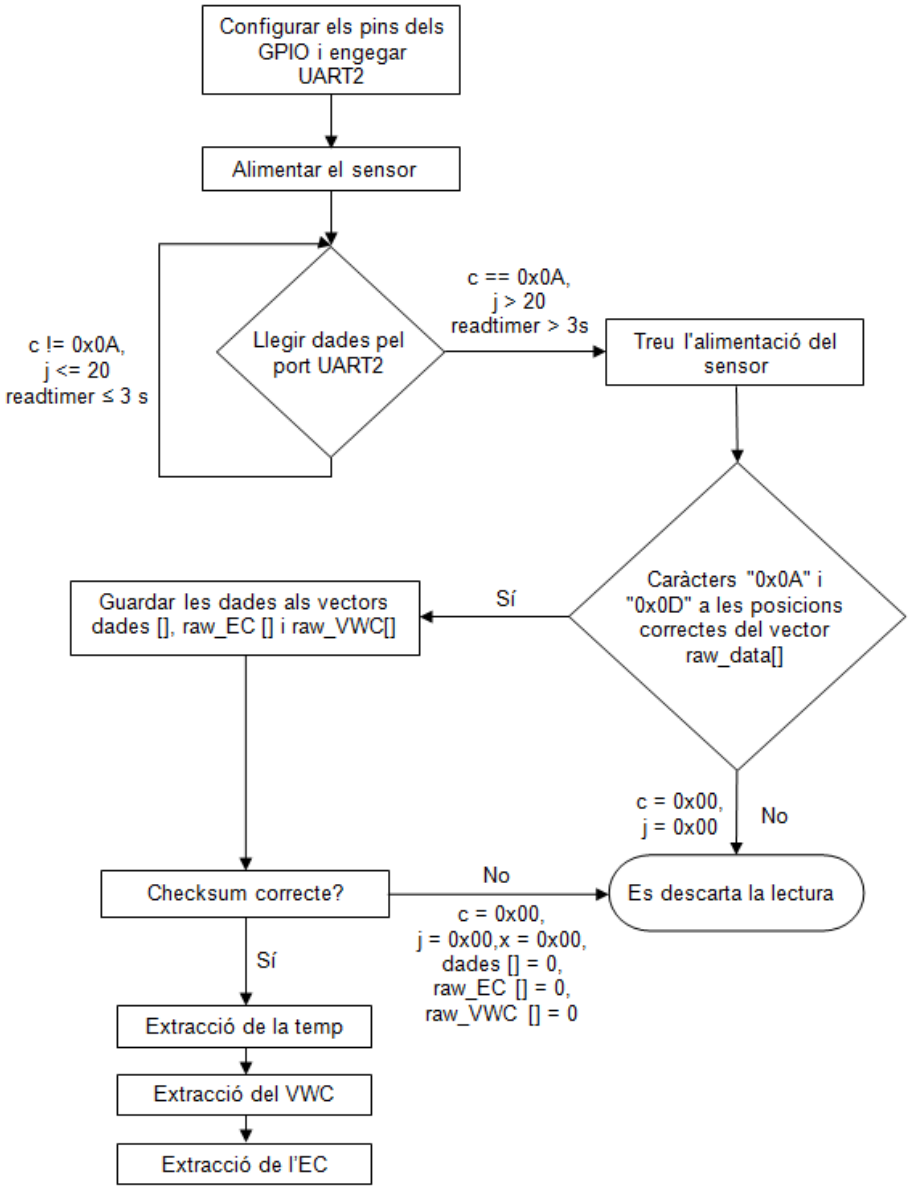

Fig. 2.8 Diagrama de l'algoritme d'operació dels sensors 5TE i 5TM.

El programa descrit al diagrama de blocs de la figura 2.9 s'ha basat en la documentació (veure referències bibliogràfiques [2] i [3]) que proporciona el fabricant per operar els sensors i gestionar la informació. Comença configurant els pins dels GPIO corresponents al mòdul UART2 i al pin que controla el MOSFET_N, i també engega el mòdul UART2. Acte seguit, alimenta el sensor (posant a *'1'* el pin del GPIO dedicat) i el programa entra en un bucle que espera a rebre caràcters pel port UART. Per sortir d'aquest bucle hi ha tres condicions: haver llegit el caràcter especial *'0x0A'*, haver llegit 20 caràcters o haver transcorregut un temps de 3 s. Un cop el programa surt del bucle treu l'alimentació del sensor (posant a *'0'* el pin del GPIO dedicat).

Després de llegir els caràcters que envia el sensor, es guarden al vector *'raw_data[]'* i es comprova si tenen el format correcte (caràcters *'0x0A'* i *'0x0D'*

a les posicions última i penúltima respectivament). Si tenen el format correcte s'extreu la informació de cada paràmetre, si el format no és l'adequat es descarta la lectura (variable *'c',* on es guarda cada caràcter llegit, i variable *'j',* que serveix d'índex pel vector *'raw_data[]'*, posades a 0).

Quan tenim els paràmetres guardats en vectors independents (vectors *'dades[]'*, *'raw_EC[]'* i *'raw_VWC[]'*) es calcula el *checksum* per verificar que les dades rebudes no s'han corromput. Si el valor obtingut coincideix amb el valor corresponent de la casella del vector de dades, s'extreu cada paràmetre (TEMP, EC i VWC, del sensor 5TE i, TEMP i VWC, del sensor 5TM).

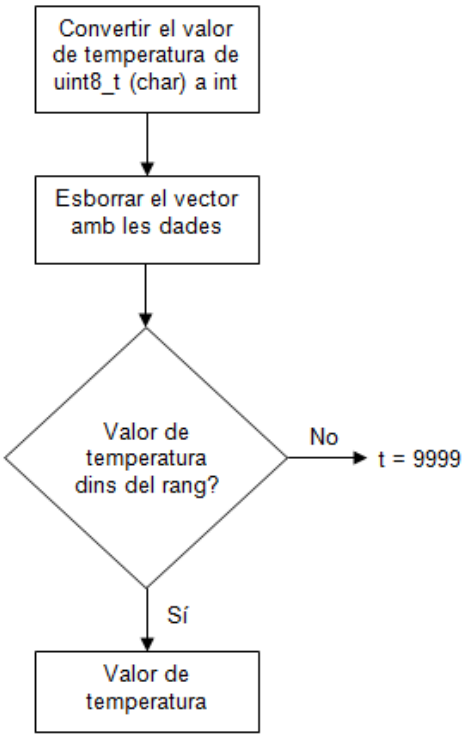

Fig. 2.9 Diagrama de l'algoritme per a l'obtenció del valor de temperatura.

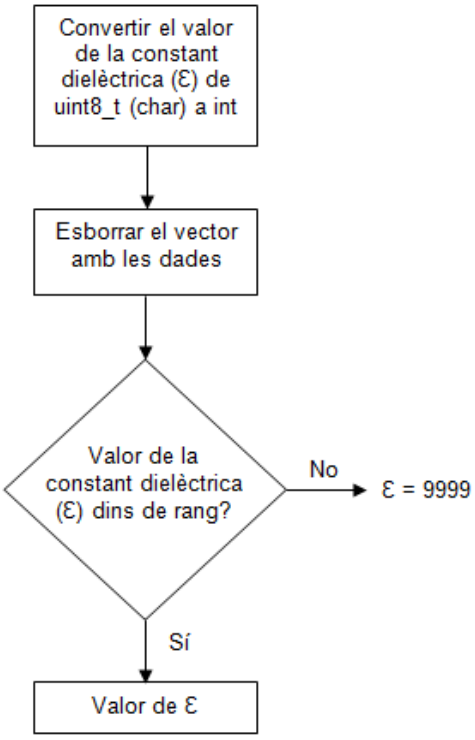

Fig. 2.10 Diagrama de l'algoritme per a l'obtenció del valor de VWC.

L'algoritme per a l'obtenció de l'EC del sensor 5TE és idèntic a l'algoritme per a l'obtenció de la temperatura. Es converteix el valor obtingut de la lectura, de tipus *uint8_t (char)* a tipus *int*. Es comprova si el valor es troba dins de rang i s'extreu el valor d'EC, del contrari es retorna el valor 9999 per indicar que el valor no està dins de rang.

A l'annex XIII d'aquesta memòria es pot revisar el mètode utilitzat per comprovar si es llegeixen correctament les dades dels sensors.

CAPÍTOL 3. INTERFÍCIE ECONOTAG – SENSOR PT100

En aquest capítol es veurà el disseny de la interfície d'interconnexió del sensor PT100 amb l'Econotag. Primer, es realitza una presentació de l'escenari per situar al lector. Després, es mostra el desenvolupament HW i SW que s'ha dissenyat per tal d'operar el sensor des de la plataforma Econotag.

3.1. Interconnexió Econotag – sensor PT100

Com s'ha descrit a la presentació dels dispositius emprats per la mesura de les magnituds requerides, el sensor PT100 és un sensor analògic. Aquest es comporta com una resistència que varia el seu valor segons la temperatura a la qual es troba el sensor.

La interconnexió d'aquest sensor amb la plataforma Econotag s'efectua segons l'esquema de la figura 3.1.

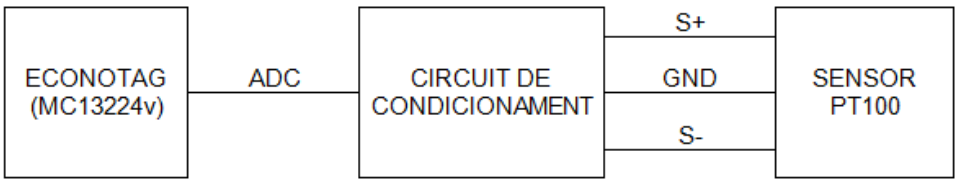

Fig. 3.1 Esquema de connexió del sensor PT100.

Un cop revisada la documentació del sensor (veure referència bibliogràfica [8]) sabem que la lectura de les dades que ens proporciona el sensor l'haurem de fer mitjançant el mòdul ADC.

A l'annex IV d'aquesta memòria es pot revisar el procediment per operar el mòdul ADC de l'Econotag.

Tal i com es pot veure a l'esquema de la figura 3.1, el sensor PT100 proporcionat disposa de tres cables per connectar-lo, dos de senyal i un de massa. Aquest disseny s'utilitza per evitar l'error en la mesura degut a la resistència del propi cablejat, i també, per evitar l'error introduït per una longitud diferent en els cables de connexió.

El mètode per obtenir la mesura del sensor és aplicant un corrent molt petit (~ 1 mA) per tal de mesurar la caiguda de tensió al transductor RTD. Aquest corrent ha de ser petit per no auto escalfar el sensor i provocar un error en la mesura obtinguda.

3.2. Circuit de condicionament del senyal

El circuit proposat per a la mesura de la temperatura amb el sensor PT100 és una petita variant del pont de Wheatstone[17]. Els requeriments inicials d'aquest sistema són:

- V_{in} màxima del xip mc13224v (Econotag): V_{SS} (GND) + 0.2 V < V_{BATT} (~ 3.3 V_{cc}) - 0.2 V.
- Alimentació del sensor: +5 Vcc.
- Mínim consum.

A la figura 3.2 es pot veure el disseny del circuit.

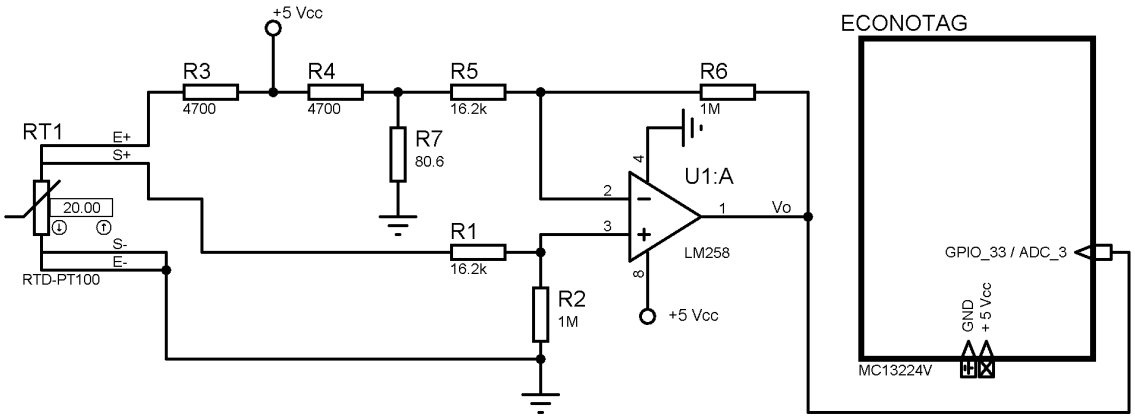

Fig. 3.2 Circuit d'adaptació del senyal del sensor PT100.

Tenint en compte els requeriments de precisió en el valor de resistència anteriorment descrits, les resistències tenen un valor de tolerància de l'1%. Per escollir l'amplificador operacional s'ha tingut en compte el consum i els nivells de tensió necessaris per al seu funcionament. Com a resultat, el model escollit per realitzar l'amplificació diferencial del nostre sistema de condicionament del senyal és el model LM258 de Texas Instruments (veure referència bibliogràfica [9]).

Si s'observa el circuit d'adaptació de la figura 3.2 es pot veure que aquesta RTD és de 4 fils i no de 3 com la que disposem. Després de revisar el datasheet del sensor s'ha resolt que connectant els fils *"S-"* i *"E-"* a massa i utilitzant els fils *"E+"* i *"S+"* com els cables de senyal, el diagrama de connexió queda igual que amb un sensor de 3 fils. Per tant, podem utilitzar aquest sensor al simulador de circuits Proteus per acabar d'ajustar els valors de les

[17] **Pont de Wheatstone.** És un circuit utilitzat per mesurar resistències que tenen un valor desconegut mitjançant l'equilibri dels braços del pont. Disposa d'una resistència variable per tal d'ajustar la relació entre resistències d'un dels braços del pont.

resistències perquè la tensió de sortida (V_o) s'ajusti als llindars de tensió de l'ADC de l'Econotag coneixent el marge de temperatures què hem de mesurar.

A l'annex VI d'aquesta memòria es pot revisar la placa prototipus que s'ha muntat, amb els components del circuit de condicionament soldats.

3.3. Lectura i processat de les dades

Per programar el codi que definirà la llibreria del sensor PT100 es realitza un diagrama de blocs que defineixi l'algorisme de lectura i processat de les dades provinents del sensor. Aquest algorisme permetrà organitzar el codi quan programem, així com tenir-lo correctament estructurat. El diagrama de blocs definit per a l'adquisició i el processament de les dades és el de la figura 3.3.

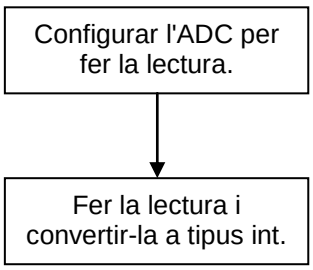

Fig. 3.3 Diagrama de blocs de l'algoritme de maneig del sensor PT100.

Després de programar el codi per llegir el senyal del sensor i obtenir la temperatura necessitem trobar la corba de calibratge del sensor. Per fer-ho es realitzen lectures periòdiques amb el sensor PT100 i un altre sensor de referència. Aquestes lectures han de ser en el mateix instant de temps i en un medi que compensi al màxim l'inèrcia tèrmica dels sensors per tal de no introduir un error (veure annex V d'aquesta memòria). Per últim, guardem aquestes mesures per al seu posterior anàlisi i disseny de la corba de calibratge del sensor PT100.

En el nostre cas, el sensor de referència utilitzat és el sensor 5TM ja comentat anteriorment. S'utilitza aquest sensor perquè té una precisió suficient com per utilitzar-lo de patró de mesura.

Quan s'han obtingut les mesures d'ambdós sensors en tot el rang de temperatures requerit, passem a representar la corba de calibratge del sensor. Per fer-ho, necessitem trobar una recta, o conjunt de rectes, que s'aproximin el màxim possible a la corba del sensor 5TM perquè, com hem comentat, aquest sensor té una precisió suficient com per utilitzar-lo de patró de mesura.

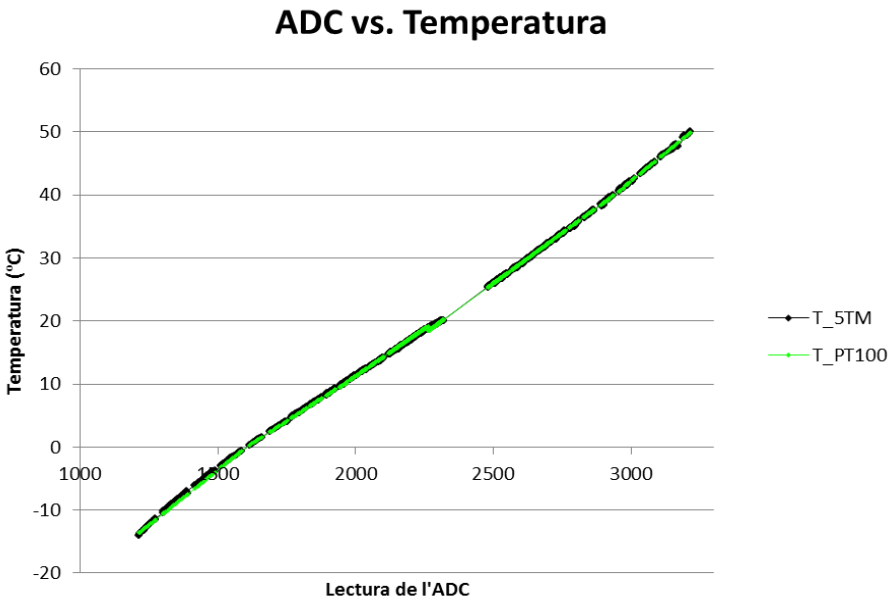

Fig. 3.4 Lectures dels sensors 5TM i PT100.

A la figura 3.4 es pot observar com les lectures del sensor PT100 segueixen la mateixa tendència que les lectures del sensor 5TM en quasi tot el rang.

Per calcular quin és l'error (en graus Celsius) present en cada punt utilitzem les fórmules següents:

$$T_{5TM} = T_{PT100} - \text{Error } (\varepsilon)$$

$$\text{Si } T_{5TM} > T_{PT100}, \quad \varepsilon = -\,||T_{5TM}| - |T_{PT100}||$$

$$\text{Si } T_{5TM} < T_{PT100}, \quad \varepsilon = ||T_{5TM}| - |T_{PT100}||$$

Els resultats del càlcul de l'error permetran acabar d'ajustar la corba de calibratge correctament.

Finalment, la corba de calibratge del sensor PT100 té 5 pendents diferents. Les funcions utilitzades en cada rang concret de temperatures són les de la taula 3.1. S'han escollit aquestes corbes perquè són les que presenten un marge d'error més baix en tot el rang de temperatures requerit.

Taula 3.1 Funcions de la corba de calibratge del sensor PT100.

ADC (temperatura (ºC))	FUNCIÓ (y = m · x + b)
0 (-15) < T ≤ 1587 (-0.6)	m = 0.034883721; b = -55.96046512;
1587 (-0.6) < T ≤ 1942 (9.6)	m = 0.028527607; b = -45.8006135;
1942 (9.6) < T ≤ 2263 (18.8)	m = 0.03017656501; b = -49.09919743;
2263 (18.8) < T < 2978 (41.6)	m = 0.03192771084; b = -53.84457831;
2978 (41.6) ≤ T < 3220 (50)	m = 0.035443038; b = -64.1;

A la figura 3.5 es pot veure representat l'error en graus Celsius per cada valor de temperatura.

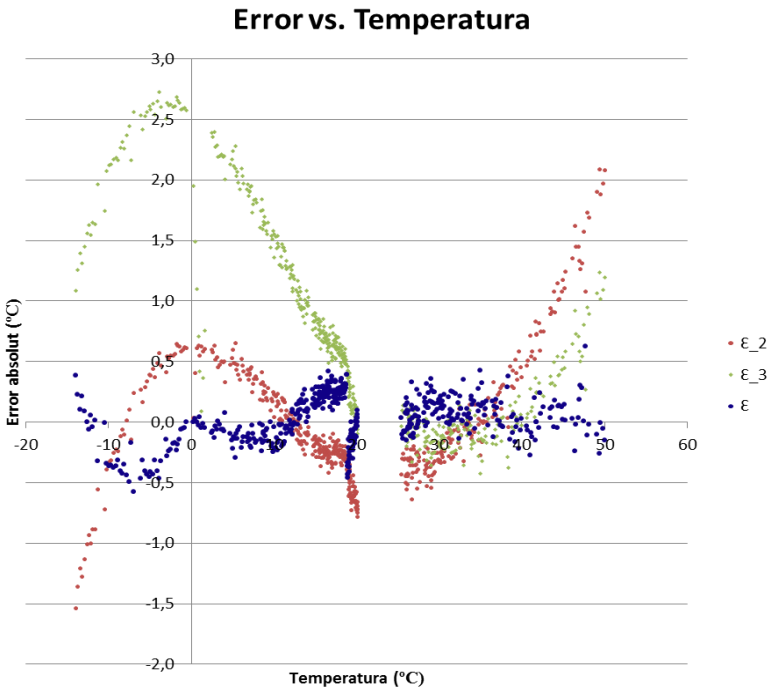

Fig. 3.5 Gràfic de l'error absolut de les corbes de calibratge.

Les corbes Ɛ_2 i Ɛ_3 corresponen a l'error introduït si utilitzem una corba fora del seu rang de temperatures, mentre que la corba Ɛ correspon a l'error introduït per la corba de cal·libració definitiva.

A l'annex XIII d'aquesta memòria es pot revisar el mètode utilitzat per comprovar si es llegeixen correctament les dades del sensor.

CAPÍTOL 4. CREACIÓ DE LA WSN

Un cop funcionen correctament les interfícies dels sensors utilitzats en el sistema, passem a crear la xarxa de sensors. Per fer-ho, primer es veuran els elements que formen la WSN, i després, es veurà la solució escollida per cadascun d'aquests elements per muntar la WSN.

4.1. Elements que formen part de la xarxa de sensors

La xarxa de sensors estarà formada per dos tipus de nodes més un tercer que permetrà comprovar el seu correcte funcionament. Aquests nodes són:

1- **Border router:** Una RPi B+ amb una Econotag (en mode slip-radio com a interfície IEEE 802.15.4). Pila TCP/IP sobre Ethernet i 6LoWPAN sobre IEEE 802.15.4.
2- **Node sensor:** Econotag amb pila 6LoWPAN i servidor CoAP. Aquest dispositiu té connectats els sensors que proporcionen els paràmetres ambientals.
3- **PC:** client CoAP (complement Copper pel navegador Mozilla Firefox) per interrogar sensors. Pila TCP/IP sobre Ethernet.

4.2. Border Router

El Border router és el sistema que farà la conversió de pila de protocols. L'explicació de la posada en marxa del BR s'ha desglossat en dos parts, SW 6LBR i configuració del sistema.

4.2.1. Software 6LBR

Al capítol 1 d'aquesta memòria s'han vist les característiques principals d'aquest SW. Ara es veurà amb més detall com funciona aquest SW adaptat a l'escenari a desplegar a l'hivernacle. Al diagrama de la figura 4.1 es mostra com funciona la solució 6LBR pel nostre cas.

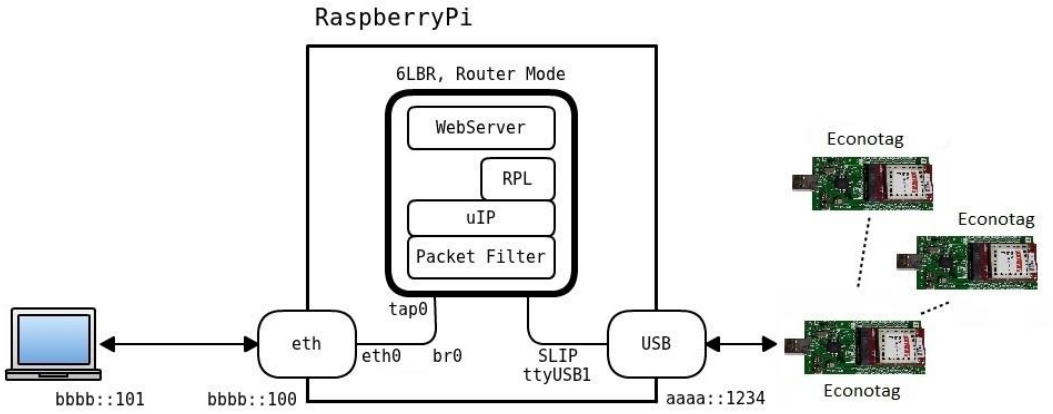

Fig. 4.1 Diagrama de l'escenari WSN amb 6LBR.

Al diagrama de la figura 4.1 veiem el mode *'router'* del SW 6LBR. En aquest mode, el SW 6LBR actua com un router IPv6. Aquest router permet connectar dues subxarxes IPv6 on, la subxarxa WSN està gestionada pel protocol RPL i, la subxarxa Ethernet està gestionada pel protocol IPv6 NDP[18].

En el mode '*router'*, el SW 6LBR proporciona una interfície virtual al SO (*tap0*) gràcies al mòdul *'Packet filter'*. També permet aïllar la WSN, i la mobilitat dels nodes entre diverses WSN gràcies a la capacitat d'intercanvi dels prefixos de xarxa.

Gràcies a l'interfície virtual proporcionada pel SW 6LBR podem fer les peticions als recursos CoAP des del propi BR.

4.2.2. Configuració del BR

Els passos per configurar el BR, tant la RPi com l'Econotag, es poden revisar a l'ANNEX IX d'aquesta memòria. No obstant, s'han de remarcar els consells i precaucions següents:

- Al finalitzar la càrrega del programa '*slip-radio.bin'* a la memòria flash de l'Econotag no ens ha de retornar cap avís ni error, el contrari significarà que el programa no s'ha carregat correctament i el SW 6LBR instal·lat a la RPi no s'iniciarà correctament.
- Per descarregar tot el SW necessari a la RPi ens facilitarà molt la feina que tingui connexió a Internet.

[18] **IPv6 Neighbor Discovery Protocol (NDP).** És un protocol d'Internet utilitzat a IPv6 que funciona a nivell d'enllaç. Serveix pel descobriment d'altres nodes de l'enllaç, la determinació de les adreces d'enllaç dels altres nodes, el descobriment dels routers i el manteniment de la informació sobre la disponibilitat de camins cap als altres nodes veïns actius.

- Per facilitar l'accés remot a la RPi per gestionar-la, ens facilita la feina el fet de configurar-li una IP estàtica, així ens estalviem comprovar-la cada vegada.
- Quan connectem l'Econotag via USB a la RPi, aquesta pren com a identificador al SO l' id. *'/dev/ttyUSB1'* i no l' id. *'/dev/ttyUSB0'* com seria d'esperar. Cal tenir-ho en compte quan editem l'arxiu de configuració del SW 6LBR (*6lbr.conf*).

4.3. Servidor CoAP

Quan tenim el BR preparat per funcionar és el torn de configurar el servidor CoAP a l'Econotag amb els recursos requerits. En el nostre cas publicarem com a recursos: 1 sensor 5TE, 2 sensors 5TM, 1 sensor PT100, el nivell de bateria i l'indicador LQI. Per fer-ho seguim els passos següents:

1. Definir els recursos CoAP al codi del programa del servidor.
2. Modificar l'arxiu *'project-conf.h'* dins de la ruta *'~/contiki/examples/er-rest-example/'* afegint els recursos de la plataforma. En el nostre cas seran els següents:
   - d5te (Obté la temperatura, l'EC i el VWC).
   - d5tm1 (Obté la temperatura i el VWC).
   - d5tm2 (Obté la temperatura i el VWC).
   - pt100 (Obté la temperatura ambiental).
   - lqi (Obté l'indicador de qualitat de l'enllaç ràdio, en anglès Link Quality Indicator) .
   - batt (Obté el nivell de tensió de la bateria).
3. Programar els drivers de cada sensor (arxius .c i .h). Guardar-los dins de la ruta '*~/contiki/platform/econotag/dev/'*, i declarar-los al Makefile de la plataforma ubicat dins de la ruta '*~/contiki/platform/econotag/'*.
4. Programar el codi de cada recurs dins del codi del programa del servidor.
5. Activar els sensors i els recursos dins del procés del servidor REST/CoAP.
6. Declarar el codi del servidor al Makefile ubicat a la ruta '*~/contiki/examples/er-rest-example/*', per tal que el tingui en compte a l'hora de compilar.

4.3.1. Definició dels recursos CoAP al servidor.

Mitjançant el codi de la figura 4.2 es defineixen els recursos CoAP al codi del programa del servidor.

```
/* Activate the desired resources */
//---------------------------------------------------------------------------//
#define REST_RES_BATTERY 1
#define REST_RES_RADIO 1
#define REST_RES_D5TESENSOR 1
#define REST_RES_D5TMSENSORS 1
#define REST_RES_PT100SENSOR 1
//---------------------------------------------------------------------------//

/* Define the necessary libraries to work with the sensors */
//---------------------------------------------------------------------------//
#if defined (PLATFORM_HAS_BATTERY)
#include "dev/battery-sensor.h"
#endif
#if defined (PLATFORM_HAS_RADIO)
#include "dev/radio-sensor.h"
#endif
#if defined (PLATFORM_HAS_D5TESENSOR)
#include "dev/d5te_sensor.h"
#endif
#if defined (PLATFORM_HAS_D5TMSENSOR)
#include "dev/d5tm_sensor.h"
#endif

#if defined (PLATFORM_HAS_PT100SENSOR)
#include "dev/pt100_sensor.h"
#endif
//---------------------------------------------------------------------------//
```

Fig. 4.2 Codi per activar els recursos al servidor CoAP (arxiu *coap-server.c*).

Mitjançant les definicions del tipus '*REST_RES_RECURS 1'* de la figura 4.2 activarem els recursos CoAP al servidor. D'aquesta manera, si volem deixar de publicar algun recurs, no és necessari buscar les línies de codi de cada recurs i comentar-les.

El segon apartat del codi de la figura 4.2 inclou les llibreries necessàries per operar cada sensor, però només s'inclouran a la compilació si hem definit prèviament els sensors de la plataforma a l'arxiu *'~/contiki/examples/er-rest-example/project-conf.h'*.

4.3.2. Definició dels recursos de la plataforma.

Ara declarem els recursos de la plataforma per poder-los operar des dels recursos CoAP. Dins de l'arxiu *'project-conf.h'* ubicat a la carpeta del projecte del servidor (*~/contiki/examples/er-rest-example/*) escrivim les línies de codi de la figura 4.3. Mitjançant aquestes línies de codi indiquem a la plataforma que disposa dels sensors D5TE, D5TM1 i D5TM2, PT100, RADIO – LQI i BATERIA.

```
#define PLATFORM_HAS_D5TESENSOR 1
#define PLATFORM_HAS_D5TMSENSOR 1
#define PLATFORM_HAS_PT100SENSOR 1
#define PLATFORM_HAS_RADIO 1
#define PLATFORM_HAS_BATTERY 1
```

Fig. 4.3 Codi per declarar els recursos a l'arxiu de configuració del programa.

4.3.3. Programació dels drivers dels sensors.

Després de revisar els arxius necessaris per operar un sensor des de Contiki[19] i, entès el procediment de declaració d'un nou sensor en una plataforma per utilitzar-lo com a recurs CoAP, passem a declarar els sensors de la nostra plataforma[20].

Un cop hem creat la carpeta '*dev'* amb els permisos necessaris dins de la ruta '*~/contiki/platform/econotag/'*, el següent pas és crear els drivers de cada sensor (arxius amb extensió *.c* i *.h*) amb el codi necessari per operar el sensor. Aquests arxius també han de tenir els permisos correctes (els mateixos que la carpeta *dev*, 775), del contrari, a l'hora de compilar el programa del servidor CoAP ens retornarà un error.

A continuació es veurà com definir els drivers de cada sensor. Per fer-ho, primer es mostraran els drivers més simples i, després, els més complexos. Així, primer presentarem els drivers corresponents als sensors PT100, radio - lqi i bateria, i després presentarem els drivers corresponents als sensors 5TE i 5TM.

4.3.3.1. Programació del driver del sensor PT100.

El contingut que té l'arxiu .h del driver del sensor PT100 és el de la figura 4.4.

```
#ifndef __PT100_SENSOR_H__
#define __PT100_SENSOR_H__

#include "lib/sensors.h"          // To use the sensors_sensor structure.

extern const struct sensors_sensor pt100_sensor;

#define PT100_SENSOR_TEMP          0

void pt100_init(void);
```

[19] S'han revisat altres plataformes, ja que del xip *mc13224v* (MCU de l'Econotag) no hi havia cap exemple ni documentació de com procedir.

[20] 'econotag'. No confondre amb 'redbee' o 'redbee-econotag'.

```
static int value(int type);
static int status(int type);
static int configure(int type, int c);

#endif /* __PT100-SENSOR_H__ */
```

Fig. 4.4 Arxiu *pt100_sensor.h* del driver del sensor PT100.

Si revisem la figura 4.4, els passos que seguim per definir l'arxiu .h del driver del sensor són:

1. Incloure la llibreria 'sensors.h' per poder treballar amb l'estructura sensors_sensor de Contiki.
2. Definir una nova estructura del tipus sensors_sensor anomenada amb el nom del sensor que estem definint, en aquest cas pt100_sensor. Aquesta estructura ens permet poder cridar les funcions per gestionar el sensor.
3. Definir les constants necessàries segons el nombre de valors que retorni el sensor, en aquest cas un únic valor, i se li assigna el valor 0.
4. Definir les funcions: static int value (int type), static int status (int type) i static int configure (int type, int c).

Els passos que acabem de definir són els mínims indispensables per poder definir el driver del sensor correctament. Sense qualsevol d'ells, el compilador ens retornarà errors al intentar compilar el codi del programa.

A la figura 4.5 es pot veure el contingut que té l'arxiu .c del driver del sensor.

```
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "lib/sensors.h"            // To use the sensors_sensor structure.
#include "dev/pt100_sensor.h"
#include "config.h"                 // To use the ADC module.
#include "adc.h"

const struct sensors_sensor pt100_sensor;

enum {
  ON, OFF
};
static uint8_t state = OFF;

/*---------------------------------------------------------------------------*/
static int value (int type)
{
   adc_service();
   return (int) adc_reading[3];
}
/*---------------------------------------------------------------------------*/
static int status (int type)
{
```

```
  switch(type) {
    case SENSORS_ACTIVE:
    case SENSORS_READY:
    return (state == ON);
  }
  return 0;
}
/*---------------------------------------------------------------------------*/
static int configure (int type, int c)
{
  switch(type) {
  case SENSORS_ACTIVE:
    if(c) {
      if(!status(SENSORS_ACTIVE)) {
        adc_setup_chan(3);        // Setup the desired ADC pin to read the analog value.
        state = ON;               // The adc_init() function was called from init.c file from Contiki
      }
    } else {
      state = OFF;
    }
  }
  return 0;
}
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR (pt100_sensor, "pt100", value, configure, status);
```

Fig. 4.5 Arxiu *pt100_sensor.c* del driver del sensor PT100.

Si revisem el codi de la figura 4.5 hi ha una primera part que inclou les llibreries necessàries per programar el driver del sensor. Acte seguit, es defineix l'estructura de tipus '*sensors_sensor'* anomenada '*pt100_sensor'*. Aquesta estructura permetrà operar el sensor des del recurs CoAP. També es defineixen les variables '*ON', 'OFF' i 'state'*, que més endavant s'utilitzaran a les funcions '*status()'* i '*configure()'*.

Un cop incloses les llibreries i havent definit les variables i estructura necessàries, passem a explicar les funcions que s'han definit. La primera funció '*static int value (int type)'* prepara el mòdul ADC per realitzar una mesura mitjançant la funció '*adc_service()'*.Espera un temps a què el mòdul estigui llest i retorna la lectura del mòdul ADC. La funció '*static int status (int type)'* s'encarrega d'indicar l'estat en què es troba el sensor. L'última funció '*static int configure (int type, int c)'* s'encarrega de configurar el port del mòdul ADC que realitzarà les lectures del sensor PT100 i, a més, també actualitza la variable state perquè representi l'estat actual del sensor.

Els drivers corresponents al sensor radio – lqi i al sensor del nivell de bateria són molt similars al driver del sensor PT100, pel que passarem a presentar els drivers dels sensors 5TE i 5TM. El codi complet dels drivers dels sensors radio i nivell de bateria es pot revisar a l'ANNEX X d'aquesta memòria.

4.3.3.2. Programació dels drivers dels sensors 5TE i 5TM.

Els drivers d'aquests sensors són una mica més complexos perquè comparteixen diverses funcions. Passem a veure com s'ha estructurat el codi dels diversos arxius, i quines funcions necessita cada sensor i quines són compartides entre tots els sensors.

Funcions de l'estructura *sensors_sensor:*
- *static int value (int type).*
- *static int status (int type).*
- *static int configure (int type, int c).*

Funcions dels sensors 5TE i 5TM:
- *void uart2init (void).*
- *void X_init (void).*
- *void X_on (void).*
- *void X_off (void).*
- *uint16_t Checksum (uint8_t * response, uint8_t length).*
- *void read_sensor (int sensnum).*
- *int get_temp (int sensorid).*
- *int get_vwc (void).*
- *int get_ec (void).*

Fig. 4.6 Funcions dels drivers dels sensors 5TE i 5TM.

A la figura 4.6 es poden veure totes les funcions que formen part dels drivers dels sensors 5TE i 5TM. Les funcions de l'estructura *sensors_sensor* pertanyen al SO Contiki i serveixen d'interfície entre el sensor i el SO. Les funcions de la columna de la dreta són les que s'han programat per tal de poder operar els sensors. Les funcions que estan subratllades són les que hem d'adaptar a cada sensor de la plataforma, en el nostre cas un sensor 5TE i dos sensors 5TM.

L'estructura que s'ha seguit a l'hora de crear els diferents arxius *.c* i *.h* ha estat, per una banda, ajuntar les funcions compartides pels dos tipus de sensors, 5TE i 5TM, dins dels arxius *'decagon_sensors.c'* i *'decagon_sensors.h'*, i per l'altra, crear un driver exclusiu per cada sensor *'d5tX_sensor.c'* i *'d5tX_sensor.h'*. On la X pot ser '*e', 'm1'* o *'m2'* segons quin driver estiguem mirant. A la figura 4.7 es pot veure el contingut dels diversos arxius mencionats.

Funcions compartides pels sensors Decagon:
- *void uart2init (void).*
- *uint16_t Checksum (uint8_t * response, uint8_t length).*
- *void read_sensor (int sensnum).*
- *int get_temp (int sensorid).*
- *int get_vwc (void).*
- *int get_ec (void).*

Fig. 4.7 Arxiu *decagon_sensors.c* del driver dels sensors Decagon.

A continuació es detalla la funció que desenvolupa cadascuna de les funcions de la figura 4.7:

- Funció '*void uart2init (void)'*. S'encarrega de configurar correctament el mòdul UART2 i d'iniciar-lo per poder llegir les dades dels sensors.

- Funció '*uint16_t Checksum (uint8_t * response, uint8_t length)'.* S'encarrega de calcular el *checksum* de la resposta rebuda del sensor per comprovar si la resposta conté algun error.

- Funció *'void read_sensor (int sensnum)'*. S'encarrega de realitzar la lectura del sensor, comprovar les dades, i guardar-les per al seu processat posterior. El paràmetre d'entrada '*sensnum'* s'utilitza per indicar a la funció quin sensor volem llegir.

- Funció *'int get_temp (int sensorid)'*. S'encarrega de cridar a la funció '*read_sensor (int sensnum)'* per realitzar la lectura del sensor i, posteriorment, calcular el valor de temperatura a partir del valor en brut prèviament obtingut. El paràmetre d'entrada *'sensorid'* és per indicar a la funció '*read_sensor (int sensnum)'* quin sensor volem llegir.

- Funció *'int get_vwc (void)'*. S'encarrega de calcular el valor de VWC a partir del valor en brut prèviament obtingut mitjançant la funció '*read_sensor (int sensnum)'*.

- Funció *'int get_ec (void)'*. S'encarrega de calcular el valor de EC a partir del valor en brut prèviament obtingut mitjançant la funció '*read_sensor (int sensnum)'*.

A la figura 4.8 es poden veure les funcions del driver del sensor 5TE.

Funcions pel sensor 5TE:
- *void d5te_init (void).*
- *void d5te _on (void).*
- *void d5te _off (void).*
- *static int value (int type).*
- *static int status (int type).*
- *static int configure (int type, int c).*

Fig. 4.8 Arxiu *d5te_sensor.c* del driver del sensor 5TE.

- Funció *'void d5te_init (void)'*. S'encarrega de configurar els registres del GPIO que opera el sensor.
- Funció *'void d5te_on (void)'*. Treu un *'1'* pel pin del GPIO configurat a la funció *'void d5te_init (void)'*.
- Funció *'void d5te_off (void)'*. Treu un *'0'* pel pin del GPIO configurat a la funció *'void d5te_init (void)'*.
- Funció *'static int value (int type)'*. Recupera el valor dels paràmetres del sensor (EC, VWC i temp, en aquest cas).

- Funció *'static int status (int type)'*. Mostra l'estat en què es troba el sensor (actiu, preparat per llegir o apagat).
- Funció *'static int configure (int type, int c)'*. Crida les funcions per configurar el sensor i en canvia l'estat ('ON' o 'OFF').

El codi complet dels drivers de cada sensor es pot revisar a l'annex X d'aquesta memòria.

4.3.3.3. Declaració dels drivers a la plataforma.

L'últim pas necessari per definir un sensor a la plataforma Econotag és modificar el Makefile de la plataforma (*~/contiki/platform/econotag/Makefile.econotag*) perquè tingui en compte els arxius de cada driver a l'hora de compilar el programa del servidor CoAP. A la figura 4.9 es pot veure el contingut d'aquest fitxer.

```
# -*- Makefile -*-

CONTIKI_TARGET_DIRS = . dev apps net
CONTIKI_CORE = main
CONTIKI_TARGET_MAIN = ${CONTIKI_CORE}.o

CONTIKI_TARGET_SOURCEFILES += main.c clock.c button-sensor.c sensors.c slip.c
platform_prints.c
CONTIKI_TARGET_SOURCEFILES += decagon_sensors.c d5te_sensor.c d5tm1_sensor.c
d5tm2_sensor.c
CONTIKI_TARGET_SOURCEFILES += pt100_sensor.c battery-sensor.c radio-sensor.c

${warning $(CONTIKI)}
CONTIKIMC1322X=$(CONTIKI)/cpu/mc1322x
CONTIKIBOARD=.

CONTIKI_PLAT_DEFS =

MCU=arm7tdmi-s

ifeq ($(UIP_CONF_IPV6),1)
CFLAGS += -DWITH_UIP6=1
endif

include $(CONTIKIMC1322X)/Makefile.mc1322x
```

Fig. 4.9 Arxiu de configuració de la compilació, '*Makefile.econotag'*.

Les modificacions fetes a l'arxiu de configuració de la compilació (*Makefile.econotag*) són les dues línies de codi subratllades a la figura 4.9.

4.3.4. Programació dels recursos del servidor CoAP.

Després de programar els drivers de cada sensor procedim a programar el codi dels recursos que tindrà cada servidor CoAP (arxiu *coap-server.c* ubicat a la ruta '*/home/user/contiki/examples/er-rest-example/*'). Els passos a seguir per programar un recurs al servidor són els següents:

1. Declarar el recurs mitjançant l'instància *'RESOURCE (nom del recurs, tipus de recurs (GET, POST o OBSERVE), URI del recurs, títol del recurs,etiqueta del recurs)*.
2. Declarar la funció que gestionarà les peticions al recurs que s'ha creat. Aquesta funció té el format *'nom del recurs_handler(void* request, void* response, uint8_t *buffer, uint16_t preferred_size, int32_t *offset)'*, i és on s'obtenen les dades del sensor i es gestionen per ser enviades.
3. Programar el codi necessari dins de la funció *'nom del recurs_handler(...)'* per obtenir les dades del sensor i gestionar-les adequadament per enviar-les. Dins d'aquesta funció també s'incrementa el valor de la variable global *'msgcounter'* per poder tenir un control dels missatges enviats, així com per poder controlar missatges duplicats.

A la figura 4.10 es pot veure el codi del recurs LQI.

```
RESOURCE(radio, METHOD_GET, "sensors/Radio", "title=\"Radio LQI\";rt=\"Radio LQI
Sensor\"");

void
radio_handler(void* request, void* response, uint8_t *buffer, uint16_t preferred_size, int32_t
*offset)
{

  int lqi = radio_sensor.value(0); // Make the reading of the LQI (Link Quality Indicator) value.

  lqi = ((lqi / 3) - 100);                // Following the advices in MC1322x Reference Manual page
169.

  msgcount++;     // Increment the message counter.

  /*  Get the header of the petition message received and response accordingly.  */
  const uint16_t *accept = NULL;
  int num = REST.get_header_accept(request, &accept);

  if ((num==0) || (num && accept[0]==REST.type.TEXT_PLAIN)) // If the format of the required
data is different from TEXT_PLAIN refuse the petition.
  {
    REST.set_header_content_type(response, REST.type.TEXT_PLAIN);

    snprintf((char *)buffer, REST_MAX_CHUNK_SIZE, "%d::%d dBm", msgcount, lqi);     //
Format the data and save it to the output buffer.

    REST.set_response_payload(response, buffer, strlen((char *)buffer));
  }
  else  {
    REST.set_response_status(response, REST.status.NOT_ACCEPTABLE);
```

```
    const char *msg = "Supporting content-types text/plain";
    REST.set_response_payload(response, msg, strlen(msg));
  }
}
```

Fig. 4.10 Codi del recurs CoAP 'Radio - *LQI'* (arxiu *coap-server.c*).

El codi dels altres recursos segueix el mateix patró. Primer, es llegeix el sensor. Després, es realitzen els càlculs i les comprovacions pertinents amb les dades rebudes. Per acabar, es formategen les dades adequadament per enviar-les.

El codi dels recursos restants es pot revisar al codi del servidor CoAP disponible a l'ANNEX XI d'aquesta memòria.

4.3.5. Activació dels sensors i els recursos al servidor CoAP.

Després de programar els recursos del servidor CoAP, el següent pas és activar els sensors i els recursos al propi servidor (arxiu *coap-server.c*). Això es fa al procés '*rest_server*' que s'encarrega de fer funcionar el propi servidor.

A la figura 4.11 es poden veure les línies de codi necessàries per realitzar l'activació dels sensors i els recursos al servidor.

```
SENSORS_ACTIVATE(radio_sensor);
rest_activate_resource(&resource_radio);
```

Fig. 4.11 Codi per activar un sensor i un recurs al servidor CoAP.

4.3.6. Compilació del codi del servidor CoAP.

L'últim pas abans de poder comprovar que la compilació no dóna errors és modificar el Makefile de la carpeta del projecte (*~/contiki/examples/er-rest-example/*) perquè tingui en compte l'arxiu del servidor CoAP *coap-server.c*.

Si el procés de compilació no retorna cap error, el log de sortida de l'execució del programa del servidor CoAP ha de ser com el de la figura 4.12.

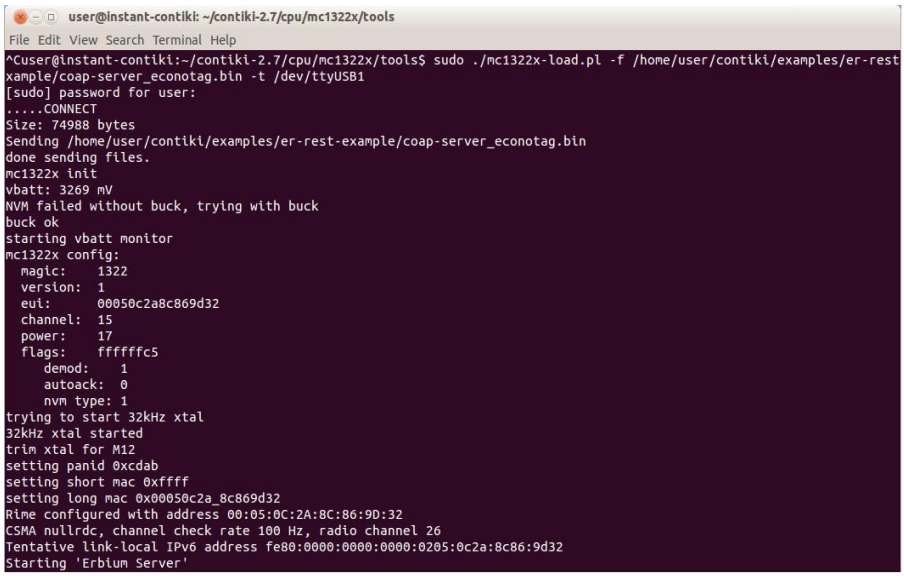

Fig. 4.12 Log d'execució del programa *'coap-server_econotag.bin'*.

4.4. Xarxa complerta

Per comprovar que el BR està funcionant correctament, un cop hem connectat l'Econotag a la RPi via USB, i la RPi a l'ordinador via el cable de xarxa Ethernet, configurem l'interfície Ethernet del PC amb els paràmetres de la figura 4.13 perquè puguem tenir connectivitat amb la RPi.

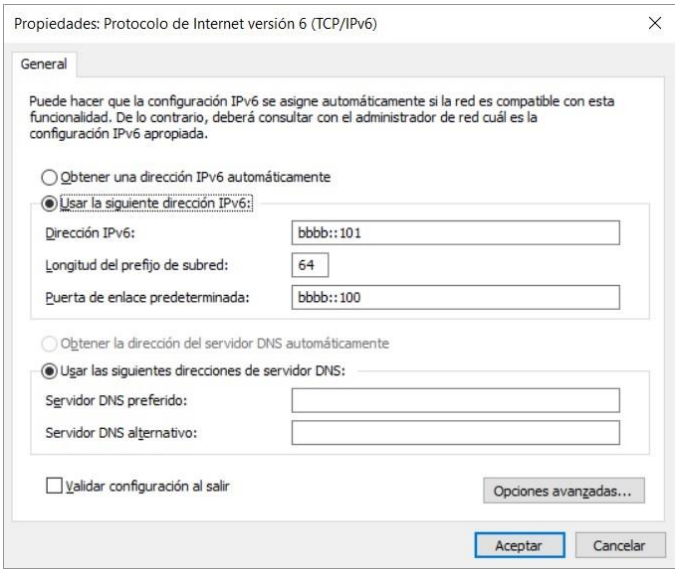

Fig. 4.13 Configuració IPv6 de l'interfície Ethernet del PC.

Iniciem una sessió SSH des del PC a la RPi mitjançant el client *'Putty'* i reiniciem el servei *6lbr* a la RPi mitjançant la comanda següent:

sudo service 6lbr restart **(4.1)**

Si no ens retorna cap error a l'hora d'executar la comanda, obrim una finestra del navegador i escrivim l'adreça *[bbbb::100]*, que és la predeterminada per accedir a la web del BR. Si el BR funciona correctament ens ha de mostrar una pàgina web com la de la figura 4.14.

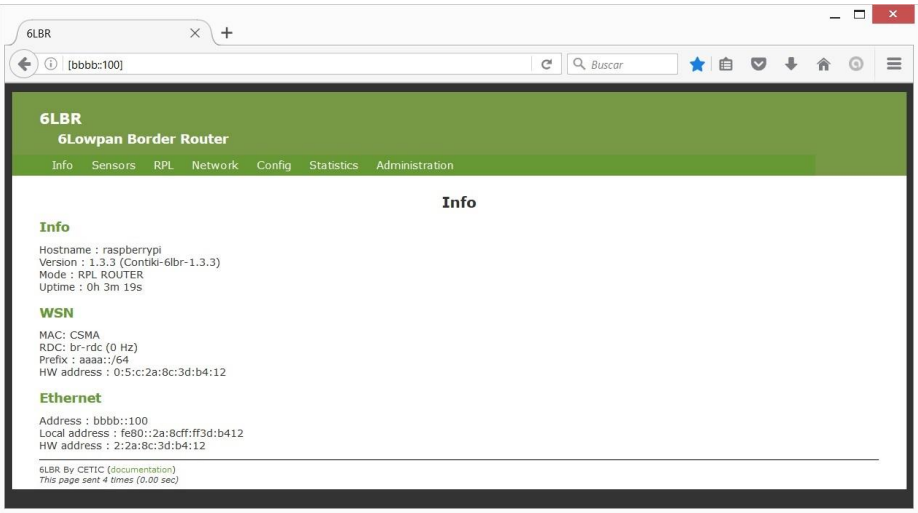

Fig. 4.14 Pàgina web inicial del BR 6LBR.

A la figura 4.14 es pot veure la pàgina inicial que ens mostra la web de gestió del BR 6LBR quan ens hi connectem. Per una banda, mostra informació del host (nom, versió del SW 6LBR, mode d'operació, temps d'operació des de l'arrencada), per l'altra, informació de les dues interfícies involucrades (Ethernet i WSN).

Després de comprovar que el servei *6lbr* està corrent a la RPi, necessitem comprovar que els diferents nodes sensors de la WSN són descoberts pel BR i, alhora, accessibles mitjançant una adreça IPv6 des de fora de la WSN, és a dir, des de la xarxa Ethernet i des d'Internet. Mitjançant la pestanya *'sensors'* (*[bbbb::100]/sensors.html*) s'obre la pàgina web de la figura 4.15.

Revisant la figura 4.15 es veu que el BR ha detectat un node *Redwire Econotag I* amb adreça IPv6 *aaaa::205:c2a:8c66:f135*.

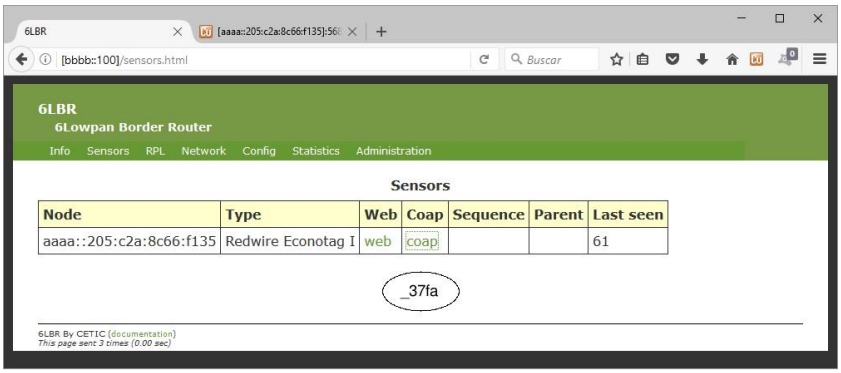

Fig. 4.15 Pàgina web [bbbb::100]/sensors.html.

L'últim pas és obrir el complement Copper del navegador Firefox (veure figura 4.16) i realitzar peticions als recursos CoAP del servidor per comprovar si es rep correctament la informació.

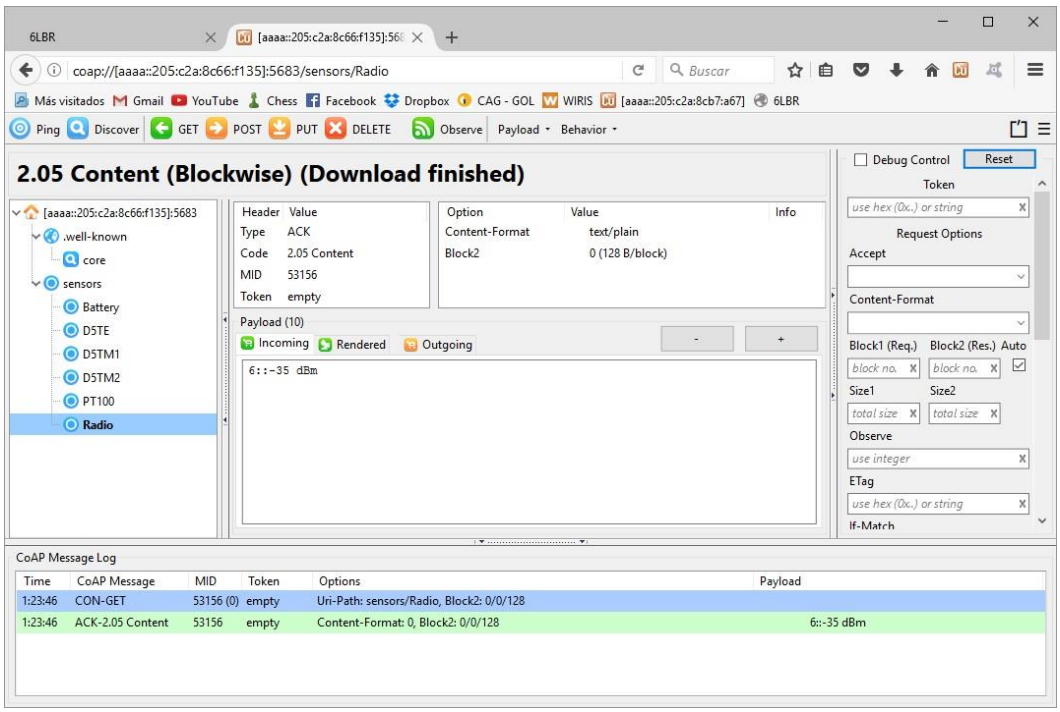

Fig. 4.16 Client CoAP. Complement Copper pel Mozilla Firefox.

El complement Copper permet seleccionar diverses opcions per formar les peticions CoAP. Per realitzar les peticions als recursos dels servidors de l'escenari s'han marcat les següents opcions de la pestanya *'Behaviour'*:

- CON requests.
- Reject Unknown.

- Block size 128.
- Observe token.
- Lazy Observe cancel.

De les opcions disponibles als camps de la dreta de la pàgina web de la figura 4.16 només s'ha marcat l'opció '*Auto',* que es troba a l'apartat '*Request Options'*.

Un cop ja funciona la xarxa complerta, passem a veure el procés d'adquisició i gestió de les dades en local a la RPi, i la posterior pujada d'aquestes dades a Internet.

CAPÍTOL 5. PUBLICACIÓ DE LES DADES A INTERNET

Gràcies a les prestacions en capacitat de processat i en memòria de la RPi, es poden realitzar les peticions als recursos CoAP des de la pròpia RPi i guardar la informació rebuda a la seva memòria local. Així, en cas de problemes a l'hora de publicar les dades a Internet tindrem una còpia de seguretat de les dades.

Inicialment, en aquest capítol es veurà com fer les peticions als recursos dels servidors CoAP des del BR, i com automatitzar aquest procés. Després, es veurà com publicar les dades dels sensors a un full de càlcul de Google Drive i com emmagatzemar les dades a la memòria local de la RPi.

A la figura 5.1 podem veure un diagrama on se situen els diferents elements involucrats.

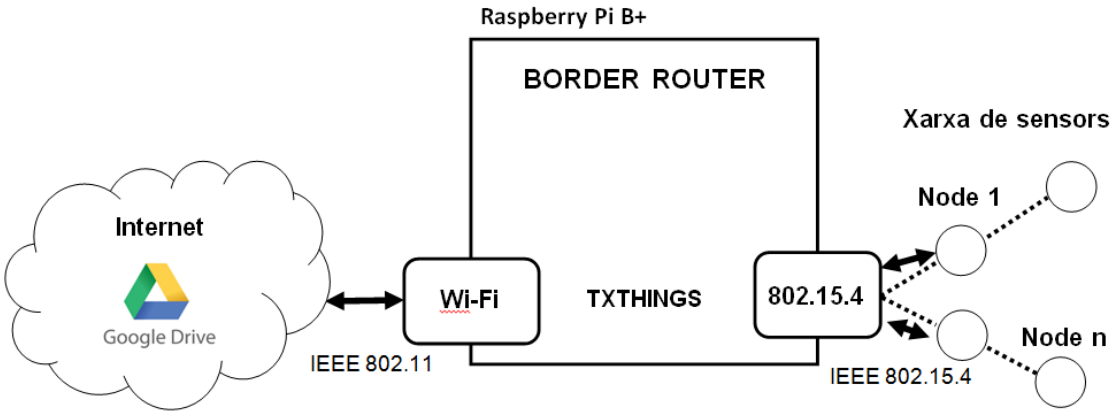

Fig. 5.1 Diagrama de la publicació de dades a Google Drive.

5.1. Petició dels recursos CoAP i automatització del procés.

Abans de poder fer les peticions als recursos CoAP des de la RPi és necessari definir una ruta IPv6 cap a la WSN, ja que el SW 6LBR no la defineix.

5.1.1. Ruta cap a la WSN des del BR.

Per definir una nova ruta a la RPi mitjançant la línia de comandes obrim una finestra del terminal i iniciem una sessió SSH des del PC a la RPi. Hem de tenir en compte que el cable Ethernet ha d'estar connectat entre el PC i la RPi.

Un cop hem iniciat la sessió SSH a la RPi, necessitem que el SW 6LBR estigui iniciat i funcioni correctament, sinó no podrem definir la ruta cap a la WSN. Per comprovar que el BR funciona correctament es poden revisar els passos de l'apartat 4.4 d'aquesta memòria.

Per establir una ruta IPv6 cap a la WSN (domini IPv6 per defecte, *aaaa::/64*) des de la pròpia RPi (adreça IPv6 per defecte, *bbbb::100*) escrivim la comanda següent:

sudo route –A inet6 add aaaa::/64 gw bbbb::100 **(5.1)**

Si el sistema no ens ha retornat cap error a l'hora d'introduir la comanda, podem revisar la ruta que hem definit mitjançant la comanda següent:

sudo route –A inet6

Un cop hem definit la ruta cap a la WSN passem a veure com realitzem les peticions CoAP des del propi BR.

5.1.2. Peticions CoAP des del BR.

Per fer les peticions als recursos CoAP des de la RPi s'utilitza un script escrit en llenguatge *Python*. Aquest script utilitza els mòduls *Twisted* i *txThings* per crear les connexions amb els servidors i construir les peticions als recursos CoAP.

A la figura 5.2 es pot veure el codi de l'script escrit en *Python*.

```
import os, time, sys
import logging

from twisted.internet.defer import Deferred
from twisted.internet.protocol import DatagramProtocol
from twisted.internet import reactor
from twisted.python import log

import txthings.coap as coap
import txthings.resource as resource

from ipaddress import ip_address

logging.basicConfig(level=logging.INFO)


class Agent():
```

```
  def __init__(self, protocol):
        self.protocol = protocol
        reactor.callLater(1, self.requestResource)

  def requestResource(self):
        request = coap.Message(code=coap.GET, mtype=coap.CON)   #++
        ## Send petition to "coap://[ aaaa::205:c2a:8c49:ce3e]:5683/sensors/D5TE"
        request.opt.uri_path = ('sensors', 'D5TE')     ## resource's URI.
        ## IPv6 CoAP server address.
        request.remote = (ip_address("aaaa::205:c2a:8c49:ce3e"), coap.COAP_PORT)

        d = protocol.request(request)
        d.addCallback(self.printResponse)
        d.addErrback(self.noResponse)

  def printResponse(self, response):
        timestamp = time.strftime("[%Y-%m-%d %H:%M:%S]", time.localtime(time.time()))
        print response.remote  # To print the remote server info (IPv6 addr. and used PORT)
        print timestamp + '::' + response.payload
        reactor.stop()

def noResponse(self, failure):
        print 'Failed to fetch resource:'
        print failure
        reactor.stop()


log.startLogging(sys.stdout)
endpoint = resource.Endpoint(None)
protocol = coap.Coap(endpoint)
client = Agent(protocol)

reactor.listenUDP (0, protocol, interface = '::0')
reactor.run()
```

Fig. 5.2 Script per realitzar una petició GET al recurs CoAP *'D5TE'*.

Si revisem el codi de la figura 5.2 hi ha una primera part on s'importen els mòduls i les llibreries necessàries per executar correctament el codi de l'script. També es configura la sortida que tindrà el log de depuració en temps d'execució.

Acte seguit, es defineix una classe *'Agent()'* que inclou les següents funcions:

- *__init__ (self, protocol).* Crea un objecte '*protocol'* i crida la funció '*requestResource(self)'* amb un retard d'un segon mitjançant la crida *'reactor.callLater(1, self.requestResource)'.*
- *requestResource(self).* Crea la petició al recurs CoAP i crida les funcions *'printResponse(self, response)'* o *'noResponse(self, failure)'* segons si ha obtingut, o no, resposta del servidor CoAP.
- *printResponse(self, response).* Imprimeix la resposta obtinguda del servidor CoAP i tanca la connexió amb el servidor mitjançant la crida *'reactor.stop()'*.

- *noResponse(self, failure).* Imprimeix una frase indicant que no ha pogut aconseguir les dades del recurs. Imprimeix l'error obtingut i tanca la connexió amb el servidor.

Amb la línia de codi *'log.startLogging(sys.stdout)'* s'activa el log de depuració en temps d'execució de l'script. Concretament, el log el traurà per la finestra del terminal des d'on s'executi l'script.

Amb les línies de codi *'reactor.listenUDP(0, protocol, interface = '::0')'* i *'reactor.run()'* s'escolta el protocol UDP i s'obre un *socket* per realitzar la petició CoAP al servidor.

Els scripts per realitzar les peticions als recursos CoAP dels altres sensors, així com els recursos de nivell de bateria, i de l'indicador LQI, són idèntics exceptuant el camp *'request.opt_uri_path'* de la funció *'requestResource(self)*, on el valor serà específic de cada recurs CoAP.

Per executar l'script de *Python* per realitzar la petició a un recurs CoAP ens hem d'assegurar que el SW 6LBR funciona correctament. Introduïm la ruta des de la RPi cap a la WSN mitjançant la comanda 5.1 i ens assegurem que l'arxiu que conté el codi de l'script de *Python* té el permís d'execució.

Per executar l'script de *Python* des de la línia de comandes, disponible a la finestra del terminal, escrivim la comanda següent:

sudo python get_coap_5te.py

Si la comanda s'executa sense cap error, la sortida per pantalla ha de ser com la de la figura 5.3.

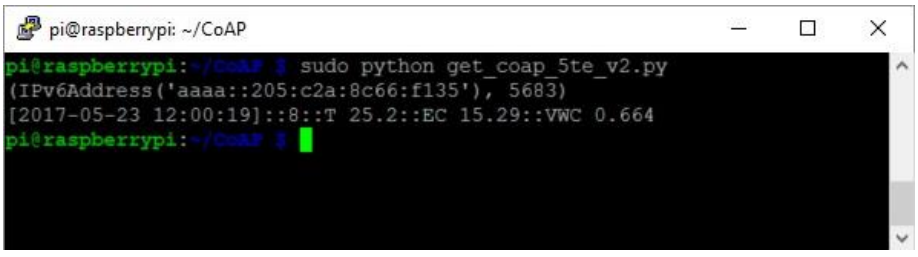

Fig. 5.3 Log de sortida de l'execució de l'script *'get_coap_5te.py'*.

Si revisem la figura 5.3, a la primera línia apareix l'adreça del servidor CoAP remot i el port utilitzat, i a la segona línia apareixen les dades rebudes del servidor amb l'etiqueta de la data i hora en què s'ha rebut el missatge. Cal puntualitzar que aquest log correspon a l'execució de l'script de *Python* havent comentat la línia de codi que activa el log de depuració.

Un cop es poden fer peticions a recursos CoAP des de la RPi passem a veure com automatitzar aquest procés i com emmagatzemar les dades a la memòria local de la RPi.

5.1.3. Automatització de les peticions CoAP i emmagatzematge de les dades.

Per automatitzar les peticions als recursos CoAP, disponibles als servidors remots, utilitzem el dimoni *cron*[21] de Linux. Per realitzar canvis a la taula del *cron* escrivim la comanda següent:

crontab –e

Afegim una nova tasca que s'executi cada 5 minuts i que realitzi la petició al recurs CoAP *'D5TE'* mitjançant la comanda següent:

**/5 * * * * sudo python /home/pi/get_5te.py >> /home/pi/get_5te.log*

Mitjançant la comanda anterior s'executa, amb permisos d'administrador, l'script de *Python* per realitzar la petició al recurs *'D5TE'* disponible al servidor CoAP. A més, es guarden les dades rebudes a l'arxiu de text *'get_5te.log'* ubicat a la ruta *'home/pi/'*.

Per tal que els canvis a la taula de tasques del dimoni *cron* s'apliquin correctament s'ha de reiniciar el servei mitjançant la comanda següent:

sudo service cron restart

Finalment, per automatitzar les peticions a tots els recursos CoAP s'ha utilitzat un script escrit en llenguatge *bash*[22]. Aquest script ens permet realitzar les peticions als recursos CoAP de forma seqüencial i deixant un marge de temps entre petició i petició. Aquest marge de temps és necessari degut al temps d'operació dels sensors. Per revisar el codi de l'script es pot fer a l'ANNEX XII d'aquesta memòria.

[21] **Dimoni *cron*.** És un servei de Linux que ens permet automatitzar l'execució de tasques i scripts perquè s'executin a una data i hora determinada.

[22] **Llenguatge *bash*.** És un programa informàtic que té per funció interpretar ordres i que s'utilitza molt sovint en sistemes GNU/Linux. S'encarrega de comunicar l'usuari amb el sistema mitjançant la línia de comandes i també mitjançant la creació de petits scripts.

5.2. Pujada de les dades a Internet.

Després d'automatitzar el procés de realitzar les peticions als diferents recursos CoAP i d'emmagatzemar les dades a la memòria local de la RPi, l'últim pas és pujar aquestes dades a Internet. L'opció escollida ha estat pujar les dades a un full de càlcul de Google Drive per facilitar el seu posterior anàlisi.

5.2.1. Plataforma Google Drive.

La plataforma Google Drive permet treballar amb diverses aplicacions en línia (editor de text, editor de fulls de càlcul, editor de presentacions, etc.), aquesta plataforma també permet accedir a documents pujats al núvol i treballar conjuntament sobre un mateix arxiu des de diferents plataformes (PC, smartphone, tablet, etc.). Aquestes característiques permeten organitzar les dades obtingudes dels sensors i visualitzar-les d'una manera més clara.

Per poder pujar les dades a un full de càlcul de la plataforma Google Drive es necessari instal·lar uns paquets *Python* al BR, esmentats a l'apartat 1.3.2 d'aquesta memòria, i configurar-lo adequadament. Per altra banda, també es necessita crear un nou projecte dins d'aquesta plataforma per poder-la enllaçar amb la nostra aplicació.

En els següents apartats es descriurà com treballar amb la plataforma Google Drive i el procés per configurar el BR. També es veurà el procés de pujada a Internet de les dades dels diferents recursos CoAP des del BR.

5.2.1.1. Creació del projecte a la plataforma Google Drive.

La creació d'aquest projecte permet enllaçar la nostra aplicació amb la plataforma Google Drive. Per crear-lo accedim a la consola de desenvolupadors de Google seguint l'enllaç web *https://console.developers.google.com/cloud-resource-manager* i seguim els passos següents:

1. Iniciem una sessió amb un compte de Google o en creem un si no en tenim cap.
2. Creem un nou projecte anomenat '*tfg-hivernacle'*.
3. Accedim a la biblioteca d'administració d'APIs mitjançant el menú desplegable de l'esquerra de la pàgina web i busquem i triem l'API '*Google Drive API'*.
4. Habilitem l'API i la consola de desenvolupadors ens demana crear unes credencials per poder-la utilitzar.
5. Creem les credencials. Ens saltem els passos i escollim l'opció '*compte de servei'* directament.
6. Definim el nom del compte de servei *'management'* i habilitem la funció *'editor'*.

7. Creem la clau privada mitjançant el botó d'opcions de la dreta i descarreguem un fitxer en format *.json*.

Un cop creat el projecte i obtingut el fitxer *.json* amb la clau privada, copiem aquest fitxer a la ruta dins de la RPi on es troba l'script de *Python* que realitza les peticions als recursos CoAP i puja les dades a Google Drive. Aquest fitxer permet autenticar la RPi a la plataforma de Google perquè pugui pujar les dades al full de càlcul de Google Drive.

L'últim pas és compartir el full de càlcul ubicat a la plataforma amb el compte de servei *'management'* que hem creat. Si no ho fem, l'script de *Python* que realitza les peticions als recursos CoAP i puja les dades a Internet ens retornarà un error dient que no pot accedir al full de càlcul.

5.2.2. Instal·lació dels mòduls necessaris al BR.

Per obtenir les dades dels recursos CoAP i pujar-les a Internet s'utilitza un script escrit en llenguatge *Python* que es veurà en els següents apartats. Perquè aquest script pugui accedir a la plataforma de Google Drive i pugui operar dins del full de càlcul és necessari instal·lar els següents paquets a la RPi:

- Gdata-python-client v2.0.18.
- Gspread v0.6.2.
- Oauth2client v4.1.0.

Per instal·lar els paquets mencionats a la RPi s'utilitza *pip*[23]. Mitjançant les comandes següents instal·lem els paquets a la RPi:

sudo pip install gdata

sudo pip install gspread

sudo pip install oauth2client

La instal·lació dels paquets mencionats requereix la instal·lació de paquets addicionals. Aquests seran requerits per la pròpia instal·lació i l'únic que haurem de fer és acceptar quan ens ho demani.

Després d'instal·lar els mòduls necessaris a la RPi passem a veure el procés per pujar les dades a la plataforma Google Drive.

[23] **Pip.** És un sistema de gestió de paquets que s'utilitza per instal·lar i gestionar paquets software escrits en llenguatge Python.

5.2.3. Pujada de les dades a Google Drive.

Per pujar les dades obtingudes dels recursos CoAP al full de càlcul de Google Drive s'han de comprovar els següents punts:

1. Ruta des de la RPi cap a Internet.
2. Data i hora correctes a la RPi.
3. Servidors CoAP funcionant.
4. BR funcionant.
5. Ruta des del BR cap a la WSN.

Si no es comprova que qualsevol dels punts anteriors està correcte, la pujada de dades al full de càlcul de Google Drive fallarà.

El codi complet de l'script per fer les peticions als recursos CoAP i pujar les dades rebudes al full de càlcul de Google Drive es pot revisar a l'annex XII d'aquesta memòria.

Un cop ja tenim els scripts preparats, l'últim pas és comprovar el funcionament del sistema complet. A l'annex XIII d'aquesta memòria es pot revisar el mètode utilitzat per comprovar si es pugen correctament les dades obtingudes al full de càlcul.

CAPÍTOL 6. CONCLUSIONS

Les conclusions d'aquest projecte és que s'han aconseguit complir tots els objectius i requeriments marcats inicialment.

Primer, s'ha realitzat el disseny de les interfícies dels sensors a nivell HW i SW i s'ha comprovat el seu funcionament. Segon, s'ha posat en funcionament una RPi com a BR i la xarxa de sensors amb els diferents nodes que la formen. Tercer, s'han aconseguit publicar els recursos HW externs a l'Econotag (sensors Decagon 5TE i 5TM, i sensor PT100) com a recursos CoAP, accessibles des de qualsevol punt amb accés a Internet mitjançant la xarxa de sensors que s'ha implementat.

Per últim, s'ha automatitzat el procés d'adquisició dels paràmetres de temperatura i humitat establerts inicialment, així com l'emmagatzematge de les dades a la memòria local del BR i en un full de càlcul de la plataforma Google Drive disponible online.

6.1. Propostes de millora i línies futures

Les propostes o línies futures per millorar aquest sistema i per seguir aquest projecte serien, per una banda, millorar la seguretat de les comunicacions a la xarxa de sensors xifrant les dades dels enllaços IEEE 802.15.4. Per altra banda, seria interessant mesurar el consum dels nodes sensors i també mirar d'optimitzar la seva configuració per disminuir aquest consum al màxim. Després, es podria estimar la seva autonomia si s'alimenten a bateries i valorar la necessitat d'implementar un sistema d'alimentació a bateries amb recàrrega mitjançant plaques solars.

També caldria muntar els nodes sensors i el BR dins de caixes amb un nivell de protecció adequat per poder ser instal·lades en l'entorn de l'hivernacle.

Una altra proposta per millorar aquest projecte seria utilitzar les dades recollides pels sensors per utilitzar actuadors que permetin millorar les condicions mediambientals de dins de l'hivernacle. Per exemple, utilitzar les dades de la temperatura ambiental per controlar l'obertura i tancament de finestres, o el control de la climatització. També es podria utilitzar l'informació dels sensors col·locats al sòl pel control de les bombes de reg.

CAPÍTOL 7. IMPLICACIONS MEDIAMBIENTALS DEL PROJECTE

L'obtenció remota, i de forma automàtica, dels paràmetres de temperatura i humitat als diferents punts de l'hivernacle permet un estalvi d'energia i temps per si mateix. Això ja disminueix l'impacte sobre el mediambient, ja que estalvia haver de fer les mesures de forma local i de manera periòdica, cosa que estalvia desplaçaments a l'hivernacle. Com s'ha descrit en aquesta memòria, els nodes-sensors d'una WSN estan dissenyats per consumir el mínim possible durant el processat de la informació i durant la transmissió i recepció de dades. L'optimització del consum energètic també es reflexa directament sobre l'impacte medioambiental.

Pel disseny i muntatge de la circuiteria de les interfícies dels sensors s'ha tingut en compte la normativa vigent aplicable en aquest cas, això és adquirir els components electrònics utilitzats amb el marcatge CE i complir amb la directriu europea ROHS (Restriction of Hazardous Substances).

Pel que fa a les interfícies ràdio IEEE 802.15.4 i IEEE 802.11, s'ha tingut en compte la normativa vigent aplicable en aquest cas, això és complir amb la directiu europea en matèria de compatibilitat electromagnètica (DC 2014/30/UE).

BIBLIOGRAFIA

[1] **Adafruit.** Raspberry Pi B+ specifications. [En línia] https://cdn-shop.adafruit.com/datasheets/pi-specs.pdf. [Ultim accés 09/2017]

[2] **Decagon Devices, Inc.** Integrators Guide 5TE Sensor. [En línia] http://manuals.decagon.com/Integration%20Guides/5TE%20Integrators%20Guide.pdf. [Ultim accés 09/2017]

[3] **Decagon Devices, Inc.** Integrators Guide 5TM Sensor. [En línia] http://manuals.decagon.com/Integration%20Guides/5TM%20Integrators%20Guide.pdf. [Ultim accés 09/2017]

[4] **NXP.** mc13224v datasheet. [En línea] http://cache.nxp.com/docs/en/data-sheet/MC1322x.pdf. [Ultim accés 09/2017]

[5] **Fairchild Inc.** 2N7000 datasheet. [En línia] https://www.fairchildsemi.com/datasheets/2N/2N7000.pdf. [Ultim accés 09/2017]

[6] **International Rectifier.** IRF4905 datasheet. [En línia] https://www.infineon.com/dgdl/irf4905.pdf?fileId=5546d462533600a4015355e32165197c. [Ultim accés 09/2017]

[7] **Fairchild Inc.** 1N4007 datasheet. [En línia] http://www.onsemi.com/pub/Collateral/1N4001-D.PDF. [Ultim accés 09/2017]

[8] **Reissmann.** PT100 datasheet. [En línia] http://www.reissmann.com/fileadmin/templates/_media/produkte/pdf/st_pt_100_en.pdf. [Ultim accés 09/2017]

[9] **Texas Instruments.** LM258-N datasheet. [En línia] http://www.ti.com/lit/ds/slos068u/slos068u.pdf. [Ultim accés 09/2017]

[10] **Contiki OS.** [En línia] http://www.contiki-os.org/. [Ultim accés 09/2017]

[11] **MC13224V Reference Manual.** [En línia] https://code.google.com/archive/p/open-mc13224v/downloads. [Ultim accés 09/2017]

[12] **Library libmc1322x.** [En línia] https://github.com/malvira/libmc1322x. [Ultim accés 09/2017]

[13] **Raspberry Pi Documentation.** [En línia] https://www.raspberrypi.org/documentation/. [Ultim accés 09/2017]

[14] **Linux man pages.** [En línia] https://linux.die.net/man/. [Ultim accés 09/2017]

[15] **CETIC 6LBR.** [En línia] https://github.com/cetic/6lbr/wiki. [Ultim accés 09/2017]

[16] **Twisted Framework.** [En línia] https://twistedmatrix.com/trac/. [Ultim accés 09/2017]

[17] **Txthings. Coap library for Twisted.** [En línia] https://github.com/mwasilak/txThings. [Ultim accés 09/2017]

[18] **Google Spreadsheets Python API.** [En línia] https://github.com/burnash/gspread. [Ultim accés 09/2017]

[19] **OAuth2Client Python library.** [En línia] https://github.com/google/oauth2client. [Ultim accés 09/2017]

ANNEX I – PROGRAMACIÓ DELS NODES ECONOTAG

Per poder programar els nodes Econotag necessitem preparar un entorn de programació que està composat dels següents elements:

- PC capaç d'executar amb fluïdesa la màquina virtual.
- Màquina virtual VMWare player versió 12.5.1.
- Entorn de programació Instant Contiki versió 2.7.

Utilitzem la versió 2.7 de l'entorn Instant Contiki, perquè el fabricant de les Econotags dóna suport fins aquesta versió. Aquest entorn és una màquina virtual amb la distribució Ubuntu del SO Linux carregada. Disposa del SO Contiki i de totes les eines de desenvolupament, com ara compiladors i simuladors.

Ens podem descarregar els arxius comprimits del següent enllaç: https://sourceforge.net/projects/contiki/files/latest/download?source=files

Per realitzar una primera aproximació al HW Econotag sense utilitzar el SO Contiki tenim a la nostra disposició una llibreria anomenada *libmc1322x*. Aquesta ha sigut creada pel dissenyador de l'Econotag, Mariano Alvira, i la defineix com una llibreria, codis de test i utilitats pel xip *mc13224v*. Aquesta llibreria ens permet iniciar-nos en aquesta plataforma i provar els mòduls que farem servir més endavant per recollir i gestionar les dades dels sensors.

Per tal de poder utilitzar aquesta llibreria des del mateix entorn de l'Instant Contiki ens descarreguem la llibreria del repositori online Github. Per a fer-ho, obrim una finestra del terminal i escrivim la comanda següent:

sudo git clone https://github.com/malvira/libmc1322x.git

La comanda anterior ens crearà una carpeta en el directori on l'haguem executat. Si entrem dins del directori podrem veure un seguit de carpetes que inclouen els codis de test, les utilitats, etc.

Per tal de compilar els codis dels programes de test, en una finestra del terminal escrivim les comandes següents des de la carpeta on hem descarregat el repositori *libmc1322x*:

cd libmc1322x/tests

make BOARD=redbee-econotag

Mitjançant l'opció *BOARD* de la comanda *make* estem especificant que volem compilar els codis dels programes només per la nostra plataforma. En aquest

cas no utilitzem la placa m12, ja que revisant els arxius de la carpeta *libmc1322x/board* hem vist que l'arxiu de configuració .h que corresponia amb el HW disposat era l'arxiu *'econotag.h'* i no l'arxiu *'m12.h'*.

Per pujar els programes compilats a l'Econotag, via el port USB, per executar-los, utilitzarem *'mc1322x-load.*pl', una utilitat disponible dins de la ruta *'libmc1322x/tools'*. Si és el primer cop que l'utilitzem segurament necessitarem instal·lar uns quants mòduls primer. Ens situarem dins de la ruta mencionada i escriurem la comanda següent:

sudo aptitude install –y libdevice-serialport-perl libterm-readkey-perl

Un cop haguem instal·lat els mòduls anteriors ja podem carregar el programa a l'Econotag mitjançant la comanda següent:

sudo ./mc1322x-load.pl –f 'ruta a l'arxiu .bin' –t /dev/ttyUSB1

A la comanda anterior indiquem a l'utilitat quin arxiu volem carregar amb l'opció *–f*. Per especificar l'ubicació on carreguem el programa ho fem amb l'opció *–t*.

ANNEX II – OPERACIÓ DELS GPIO DE L'ECONOTAG

Per comprovar el funcionament dels GPIO muntarem un circuit amb un díode LED i una resistència de 470 Ω (per limitar el corrent que passa a través del LED. I < 10 mA) i intentarem encendre i apagar el LED des d'un GPIO. El circuit proposat és el de la figura II.I.

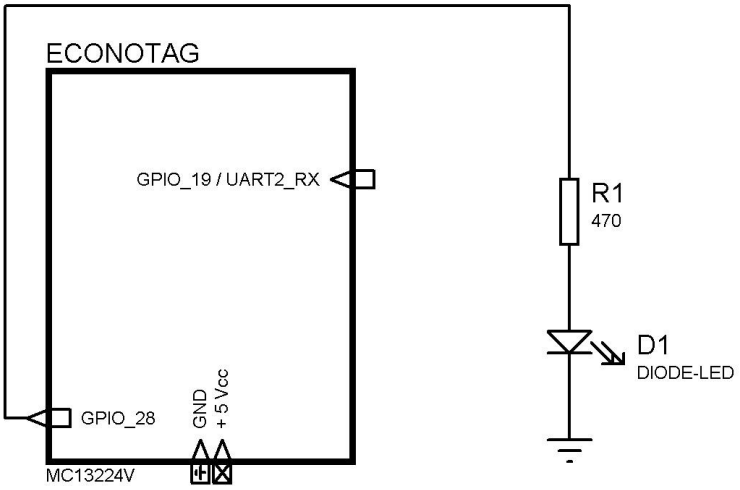

Fig. II.I Circuit per comprovar el funcionament d'un pin del GPIO com a sortida.

Per tal de poder utilitzar el pin del GPIO, del xip *mc13224v*, com a sortida s'han de seguir els passos següents:

1. Revisar que el pin del GPIO que volem utilitzar no estigui compartit amb algun altre mòdul (ADC, SPI, UART, etc.) que estigui en ús. Per fer-ho revisem el manual de referència del xip *mc13224v* que incorpora l'Econotag (veure referència bibliogràfica [11]). En el nostre cas utilitzarem els GPIO's compartits amb el mòdul KBI (de l'anglès *Keyboard Interface*).

2. Revisar els registres que s'han de modificar per seleccionar el mode de funcionament desitjat pel pin del GPIO. En el nostre cas, els registres a modificar són els següents:

   - GPIO FUNCTION SELECTION. Per escollir la funció del pin del GPIO.
   - GPIO PAD DIRECTION. Per indicar que el GPIO treballarà com a sortida.

3. Revisar la funció concreta del pin del GPIO (en el nostre cas funció 3 - Alternate).

4. Fer les modificacions dels registres pertinents (veure figura II.II) segons el pin del GPIO que haguem d'operar (posar un '0' o un '1' en la posició concreta de cada registre). És molt important que aquestes modificacions es facin sense afectar les altres posicions del registre, del contrari, podem provocar mal funcionaments del sistema, ja que estarem modificant posicions del registre dedicades a altres mòduls del xip mc13224v.

5. Assegurar que s'han fet les modificacions en els registres pertinents imprimint el valor del registre (veure figura II.III).

A la figura II.II es pot veure la modificació dels registres necessaris per operar el pin del GPIO_28 / KBI_6 com a sortida.

```
void d5tm1_init (void) {

        /* Setup the correct registers to select the desired functions of every GPIO that we are
using. */

        *GPIO_FUNC_SEL1 |= 0x3000000; // Select the function of GPIO_28 / KBI_6 (func. 3 -
alternate)
        *GPIO_PAD_DIR0 |= 0x10000000; // Setting as output the GPIO_28 / KBI_6
}
```

Fig. II.II Modificació dels registres per operar el pin del GPIO_28 com a sortida.

Si observem l'assignació del valor hexadecimal 0x3000000 (00000011000000000000000000000000 en binari) al registre GPIO_FUNC_SEL1 veiem que es fa sumant el valor en hexadecimal al valor que tenia prèviament el registre, sense afectar cap altra posició del registre que no sigui la que ens interessa. Hem de tenir en compte que els registres s'inicien per la dreta, és a dir, els dos '1' estarien en les posicions 24 i 25 del registre, respectivament, ja que el primer '0' de la dreta és la posició 0 del registre GPIO_FUNC_SEL1.

Per assegurar que els registres han estat modificats correctament i visualitzar que s'han modificat les posicions correctes podem imprimir el valor dels registres amb el codi de la figura II.III.

```
for ( i=0 ; i<=31 ; i++ ) {     /* El registre té 32 bits de 0 a 31. */

  j = ( (*GPIO_FUNC_SEL1  >> i ) & 1);

  if ( j == 1 ) {
    buffer[31-i] = '1';
  }
  else {
    buffer[31-i] = '0';
```

```
    }

  }
  printf ("GPIO_FUNC_SEL1: %s , %d", (char *) buffer, i);
  printf ("\n");
```

Fig. II.III Codi per imprimir el valor d'un registre de 32 bits.

Un cop hem comprovat que les modificacions en els registres s'han fet correctament, declarem l'arxiu del programa al Makefile corresponent (*libmc1322x/tests/makefile*) i muntem el circuit en una placa protoboard per comprovar si el programa s'executa adequadament. Si tot funciona correctament, el díode LED s'ha d'encendre i apagar de forma intermitent.

El codi del programa per comprovar el funcionament del GPIO és el de la figura II.IV.

```
#include <mc1322x.h>
#include <stdio.h>
#include <string.h>
#include "tests.h"
#include "config.h"

#define DELAY 8000000

#define PWR_ON 0x10000000
#define PWR_OFF 0xEFFFFFFF // ~on

//-------------------------------------------------------------------------------//
void gpio_init (void) {

        *GPIO_FUNC_SEL1 &= 0xFDFFFFFF; //Select the function of GPIO_28/KBI_6 (func.
1 - alternate)
        *GPIO_PAD_DIR0 |= 0x10000000; //Setting as output the GPIO_28/KBI_6

        printf("GPIO configured \n GPIO_FUNC_SEL1 = %s \n", (char*) GPIO_FUNC_SEL1);
}
//-------------------------------------------------------------------------------//
void gpio_off (void) {

  volatile uint32_t i = 0;  // Timer index
  *GPIO_DATA0 &= PWR_OFF;  // Put '0' in the GPIO_28/KBI_6
  for (i=0; i < DELAY; i++) {continue; }  // timer
}
//-------------------------------------------------------------------------------//
void main(void) {

        volatile uint32_t i = 0;    // Timer index
        printf("Initializing GPIO. \n");
        gpio_init();
        printf("GPIO initialized. \n");
        uart_init(UART1, 115200);       //Communication between the Econotag running
program and the console

        while(1) {
```

```
            printf("Inside the while. \n");
            *GPIO_DATA0 |= PWR_ON; // Put '1' in the GPIO_28/KBI_6
            for (i=0; i < DELAY; i++) {continue; }  // timer
            gpio_off();       // Put '0' in the GPIO_28/KBI_6
        }
}
```

Fig. II.IV Codi per operar un GPIO del xip *mc13224v*

ANNEX III – OPERACIÓ DEL MÒDUL UART DE L'ECONOTAG

Per comprovar el funcionament del mòdul UART, primer revisem la documentació disponible al manual de referència de la plataforma *MC1322x*. (veure referència bibliogràfica [11]), concretament el capítol 13 que està dedicat al mòdul UART. Acte seguit revisem l'exemple disponible dins de la ruta *'libmc1322x/tests'* i les llibreries corresponents al mòdul UART disponibles a la ruta *'libmc1322x/lib'*. Concretament, els arxius que ens interessen són *'u1u2-loopback.c'* com a exemple, i les llibreries *'uart1.c'* i *'uart2.c'* per entendre millor el funcionament del mòdul.

Després de revisar la documentació i el codi procedim a provar el mòdul UART mitjançant el codi de prova de la figura III.I.

```
#include <mc1322x.h>
#include <board.h>
#include <stdio.h>
#include "tests.h"
#include "config.h"

#define DELAY 6000000

void main (void) {

  char c = 0;
  volatile uint32_t i = 0;

  GPIO->PAD_PU_SEL.U2RX = 1; // Internal pull-up resistor enabled for GPIO_19/UART2_RX

  uart_init ( UART1, 115200 );   // Enable the uart modules. 1- for debbuging, 2- for reading
  uart_init ( UART2, 1200 );

  while (1) {

    printf ("Waiting data. \n");

    while ( c != 0x0A ) {

      if( uart2_can_get() ) {   /* Read from UART2 if receive any character */

        c = (char) uart2_getc();

        printf ("%02X.", c);

        if( c == 0x0A ) {  printf ("\n");  }
      }
    }

    printf ("Waiting. \n");

    for ( i=0 ; i < DELAY ; i++ ) { continue; }   // timer.

    printf ("Restart the reading process. , %ld \n", i);
```

```
  }
}
```

Fig. III.I Codi per provar el mòdul UART 2.

El procediment que segueix el codi és el següent:

1. Declaració de les llibreries necessàries per operar el xip *mc13224v* i els seus mòduls UART 1 i 2.
2. Declaració d'una variable DELAY per utilitzar-la al comptador d'espera.
3. Declaració d'una variable *'c'* per emmagatzemar els caràcters llegits pel mòdul UART2 i una altra variable *'i'* pel *timer*.
4. Configuració del registres necessaris del mòdul UART2.
5. Inicialització dels mòduls UART amb la configuració concreta per cada mòdul.
6. Creació d'un bucle infinit on llegim caràcters mitjançant el mòdul UART2 mentre puguem, i mentre el caràcter llegit no sigui el *'0x0A'*, que provocarà la finalització de la lectura. Aquests caràcters s'imprimeixen per pantalla.
7. Comptador d'espera per tornar a realitzar una altra lectura.

ANNEX IV – OPERACIÓ DEL MÒDUL ADC DE L'ECONOTAG

Per operar el mòdul ADC revisem la documentació present al manual de referència de la plataforma MC1322x (veure referència bibliogràfica [11]), concretament el capítol 17, que està dedicat a aquest mòdul. També revisem el codi corresponent. En concret l'arxiu *'adc.c'* present a les carpetes *'libmc1322x/lib'* i *'libmc1322x/tests'*. El primer arxiu conté la llibreria del mòdul ADC, mentre que el segon és un codi de test que comprova el funcionament de tots els canals del mòdul ADC.

El codi utilitzat per provar el funcionament del mòdul ADC és el de la figura IV.I.

```
#include <mc1322x.h>
#include <board.h>
#include <stdio.h>
#include "config.h"
#include "adc.h"

#define DELAY 2000000

//-------------------------------------------------------------------------------------//
void main(void)
{

  volatile uint32_t i = 0;    // Timer index

  trim_xtal();
  uart_init(UART1, 115200);
  adc_init();

  printf("adc on\r\n");

  printf("\x1B[2J");   // clear screen

  adc_setup_chan(3);

  for(;;) {
    printf("\x1B[H");   // cursor home
    printf("# Value\r\n");

    adc_service();

    printf("%u %04u %04u mV \r\n", 3, adc_reading[3], adc_voltage(3));
    printf("vbatt: %04u mV \n", adc_vbatt);

    for (i=0; i < DELAY; i++) {continue; }   // timer
  }
}
```

Fig. IV.I Codi per provar el funcionament del mòdul ADC.

Si revisem el codi de la figura IV.I es poden veure els passos necessaris per operar correctament el mòdul ADC. Els passos a seguir són els següents:

1. Configurar l'oscil·lador. Cridar la funció trim_xtal().
2. Iniciar el mòdul ADC. Cridar la funció adc_init().
3. Configurar el canal de l'ADC que utilitzarem. Cridar la funció adc_setup_channel(3).
4. Realitzar la lectura del canal de l'ADC. Cridar la funció adc_service().
5. Cridar la funció printf() amb els camps adc_reading[3] o adc_voltage(3) per obtenir el valor numèric o el valor de tensió de la conversió del canal de l'ADC que estem utilitzant.

És important no saltar-se l'ordre establert en la llista anterior (executar la funció *adc_init()* abans que la funció *trim_xtal()*), pel contrari, el mòdul ADC no funcionarà.

ANNEX V – CALIBRATGE DEL SENSOR PT100

Per realitzar el calibratge del sensor analògic PT100 s'ha hagut de tenir en compte l'inèrcia tèrmica del material que recobreix el sensor (platí).Per compensar al màxim l'inèrcia tèrmica del sensor hem enfonsat el sensor PT100 i el sensor 5TM (la seva resolució el fa apte per utilitzar-lo com a patró de mesura) dins d'un pot de vidre amb arròs. S'ha escollit l'arròs perquè manté força bé la temperatura durant l'interval entre mesures consecutives (30 s.). És important que els sensors estiguin totalment coberts d'arròs, del contrari provocarà que els sensors no estiguin a la mateixa temperatura i s'introdueixi un error en les mesures.

Per obtenir les mesures del rang de temperatures requerit hem dividit el procés en dues fases: rang inferior de temperatures (de -14 ºC a 20 ºC) i rang superior de temperatures (de 25 ºC a 50 ºC). Per realitzar les mesures del rang inferior hem posat el pot d'arròs amb els sensors dins d'un congelador a una temperatura de -18 ºC durant tot un dia. Per realitzar les mesures del rang superior hem posat el pot d'arròs amb els sensors dins del forn a una temperatura de 60 ºC durant una hora.

La realització de les mesures s'ha realitzat a intervals regulars de 30 segons fins a l'arribada a la temperatura límit dels rangs establerts prèviament.

Un cop s'han obtingut les mesures d'ambdós sensors en tot el rang de temperatures requerit, s'ha representat la corba de calibratge del sensor. Per fer-ho, es necessita trobar una recta, o conjunt de rectes, que s'aproximin el màxim possible a la corba del sensor de referència (sensor 5TM).

ANNEX VI – PLACA PROTOTIPUS

A la figura VI.I es pot veure el resultat de la placa un cop s'han soldat els components dels circuits de control de l'alimentació i dels circuits per compartir el bus de dades.

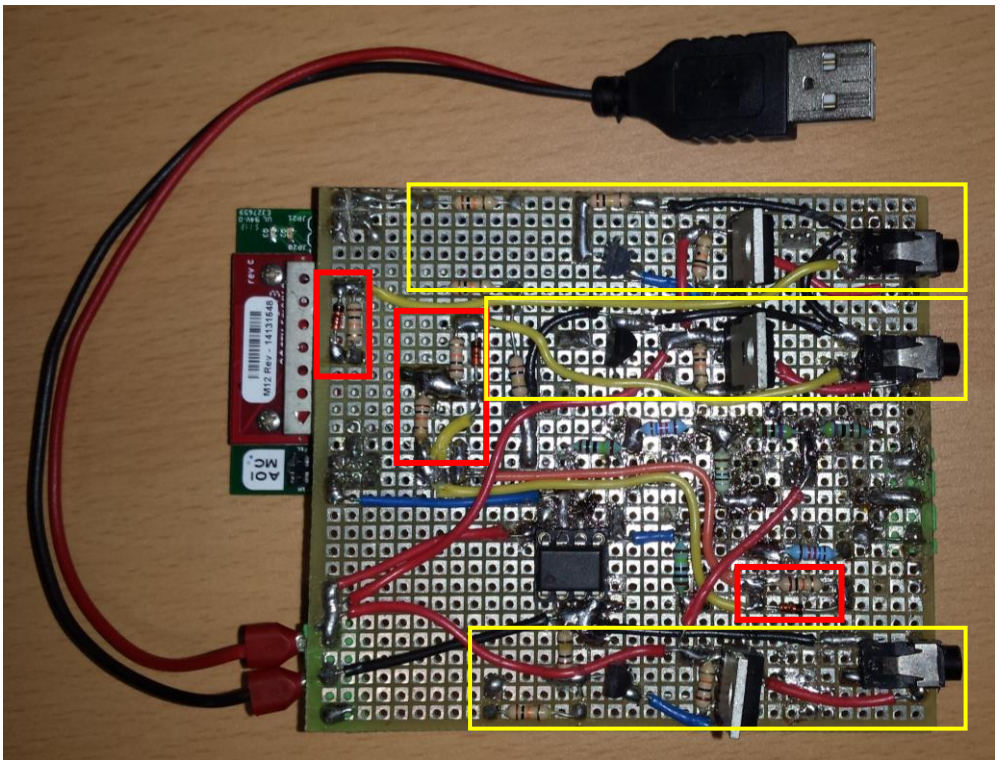

Fig. VI.I – Placa prototipus.

Els rectangles de la figura VI.I ressalten els circuits perquè sigui més fàcil identificar-los. Així, els rectangles vermells corresponen als circuits utilitzats pels sensors 5TM i 5TE per poder compartir el bus de dades (díodes i resistències de polarització). Mentre que els rectangles grocs corresponen als circuits de control de l'alimentació d'aquests sensors.

A la figura VI.II es pot veure el resultat de la placa un cop s'han soldat els components del circuit de condicionament del senyal del sensor PT100.

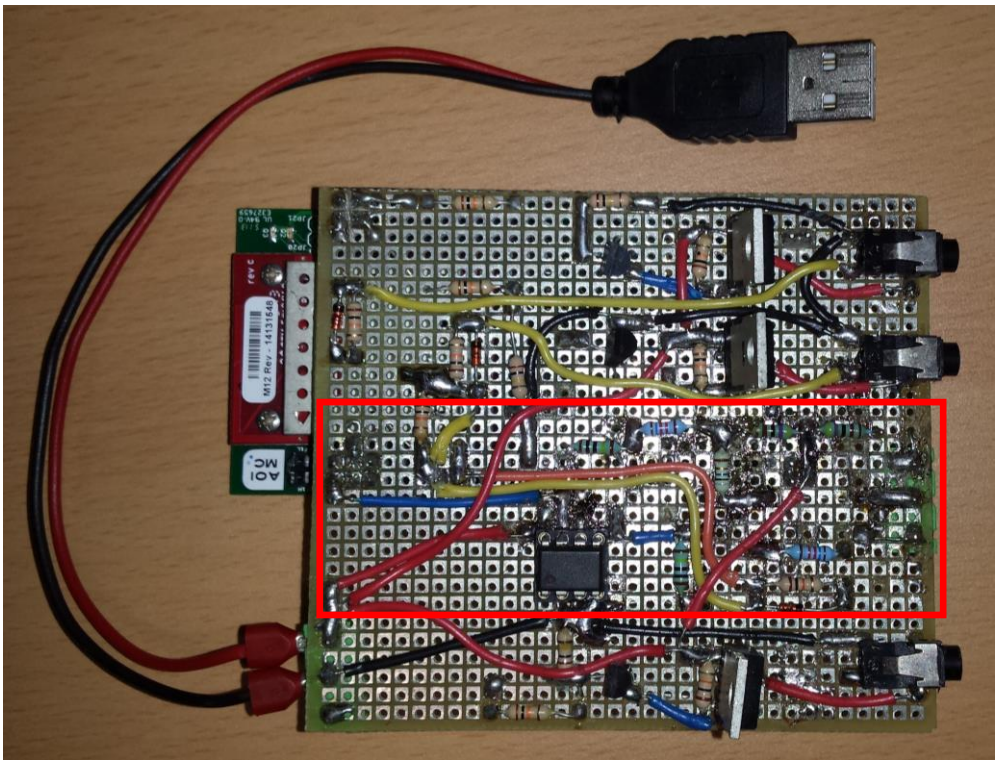

Fig. VI.II - Placa de forats amb el circuit de condicionament del sensor PT100.

A la figura VI.II, el requadre vermell ressalta el circuit de condicionament del sensor PT100. Com s'ha vist a l'apartat 3.2 d'aquesta memòria dedicat al circuit de condicionament del senyal, les resistències han de tenir una tolerància de l'1 %, que són les que tenen un color de fons verd clar i blau clar. Es diferencien clarament de les resistències amb una tolerància del 5 % que són les que tenen el color marró clar de fons. El xip integrat de vuit potes que observem és l'amplificador de baix consum *LM258*.

ANNEX VII – CIRCUIT DE CONTROL DE L'ALIMENTACIÓ

Per comprovar el correcte funcionament del circuit de control de l'alimentació de la figura 2.6 (comprovar si els transistors estan treballant correctament) canviem el sensor 5TM per un díode LED. Així, si hi ha alguna connexió mal feta o algun transistor no funciona correctament no cremarem el sensor. En aquest cas, col·locarem una resistència de valor 1 kΩ per limitar el corrent que passa a través del LED, ja que la tensió d'alimentació és de +5 Vcc. El circuit proposat és el de la figura VII.I.

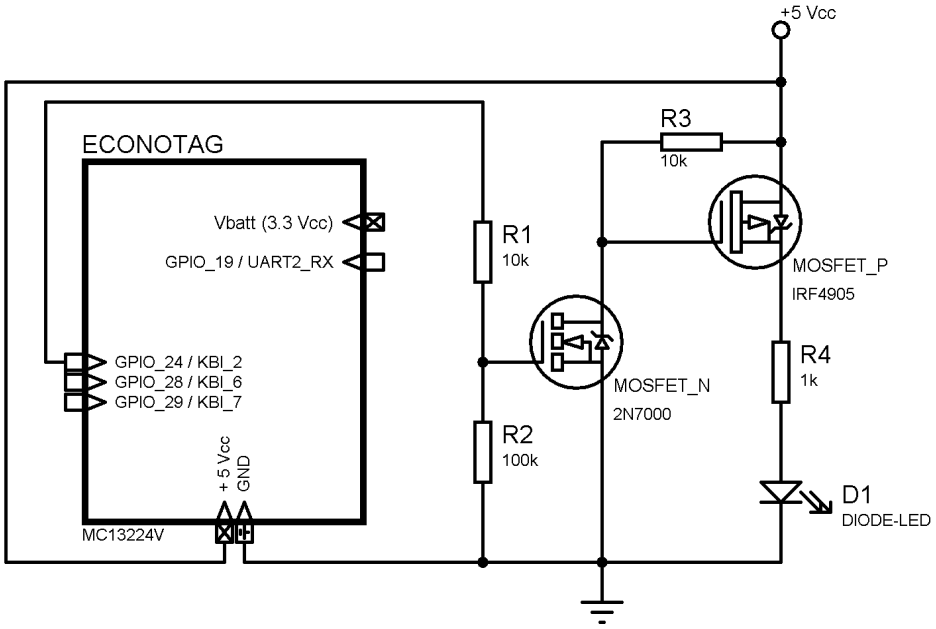

Fig. VII.I - Circuit per comprovar el control del pin del GPIO i els transistors.

Si el circuit de la figura VII.I funciona correctament, el LED s'ha d'encendre i apagar intermitentment. A l'annex II d'aquesta memòria es pot revisar el programa utilitzat per realitzar aquesta prova.

ANNEX VIII – ESTRUCTURA DE CARPETES DE CONTIKI

A la figura VIII.I es pot veure l'estructura de carpetes del SO.

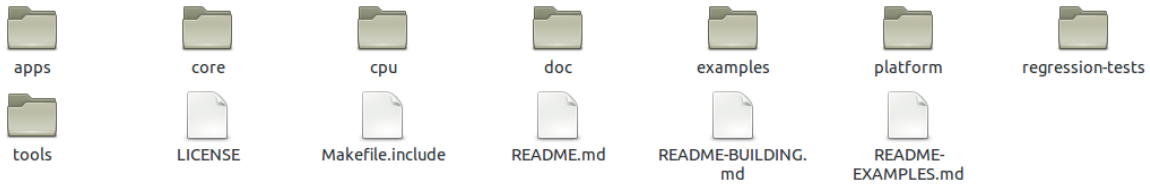

Fig. VIII.I Estructura de carpetes del SO Contiki.

El contingut de cada carpeta de la figura VIII.I es detalla a continuació:

- **apps:** En aquesta carpeta es troben les aplicacions, son programes auxiliars que es poden utilitzar al programa principal. Exemples d'aplicacions són *telnet, dhcp, ping6, webserver,* etc.
- **core:** En aquesta carpeta es troben els arxius principals del SO. Es recomana no modificar aquesta carpeta ni cap dels seus arxius. Dins d'aquesta carpeta s'hi troba l'implementació dels protothreads, de la pila uIP, del protocol RPL, i tot el que es refereix al SO.
- **cpu:** En aquesta carpeta es troba tot el codi que depèn del xip que utilitzem, en el nostre cas, el mc13224v. Dins d'aquesta carpeta s'hi troben les llibreries per operar els mòduls integrats al propi xip (ADC, UART, RADIO IEEE 802.15.4, etc.). Si es canvia de microcontrolador només s'ha de canviar de carpeta per revisar el codi corresponent al xip que utilitzem.
- **doc:** En aquesta carpeta es troba la documentació del codi del SO. També hi ha diversos exemples bàsics de funcionament.
- **examples:** En aquesta carpeta s'hi troben programes d'exemple que utilitzen el SO.
- **platform:** En aquesta carpeta s'hi troba tot el codi que depèn de la plataforma on està instal·lat el xip. Alguns exemples són la configuració dels botons i els sensors, configuració de LED's, etc. Dins d'aquesta carpeta es troba l'arxiu *main.c* (arxiu d'inicialització).
- **tools:** En aquesta carpeta es troben diverses eines extres que no formen part del SO, tals com, eines de depuració, simulació i millora de les aplicacions per a cada plataforma concreta. Per exemple, l'aplicació '*collect-view'* per la plataforma z1, el simulador COOJA, etc.

ANNEX IX – CONFIGURACIÓ DEL BORDER ROUTER

El BR d'aquest sistema està format per una unitat principal (RPi B+) més una interfície IEEE 802.15.4 (placa Econotag treballant com a '*slip-radio'*), ja que la RPi no disposa d'aquesta interfície de fàbrica.

Per configurar el BR, primer s'ha de carregar el programa *'slip-radio.bin'*, ubicat a la ruta *'home/user/contiki-2.7/examples/ipv6/slip-radio/'*, a la memòria flash d'una placa Econotag. Ens hem d'assegurar que el programa està correctament configurat, del contrari, el BR no serà capaç de descobrir els nodes sensors de la WSN. A la figura IX.I es pot veure la configuració que ha de tenir el programa.

```
NETSTACK_CONF_MAC     csma_driver
NETSTACK_CONF_RDC     nullrdc_driver
NETSTACK_CONF_NETWORK slipnet_driver
NETSTACK_CONF_FRAMER no_framer
```

Fig. IX.I – Arxiu *'project-conf.h'* del programa *'slip-radio'*.

Un cop s'ha configurat correctament el programa, el compilem introduint la comanda següent:

make TARGET=econotag

Després de compilar el programa, si no ens ha retornat cap error, el pugem a la memòria flash de l'Econotag mitjançant la comanda de la figura IX.II.

```
sudo ./mc1322x-load.pl -f /home/user/6lbr/examples/6lbr/tools/econotag/flasher_m12.bin -s
/home/user/contiki-2.7/examples/ipv6/slip-radio/slip-radio_econotag.bin -t /dev/ttyUSB1
```

Fig. IX.II – Comanda per pujar el programa '*slip-radio_econotag.bin'* a l'Econotag.

La comanda de la figura IX.II s'ha d'executar des de la ruta *'/home/user/contiki-2.7/cpu/mc1322x/tools/'*.

Si el procés de pujada del programa a la memòria flash ha finalitzat correctament no ens ha de retornar cap avís ni error, del contrari, el programa no s'haurà pujat correctament i l'Econotag no funcionarà com a slip-radio.

El següent pas és configurar la RPi. Per fer-ho, necessitem accedir remotament a la RPi mitjançant una sessió SSH. Abans però, necessitem configurar

adequadament les adreces de les interfícies Ethernet del PC i de la RPi perquè es puguin descobrir. Les adreces utilitzades són les següents:

RPi B+: 192.168.137.2/24

PC: 192.168.137.1/24

Per tal de definir una adreça IPv4 estàtica a la RPi s'ha de modificar l'arxiu de configuració '*cmdline.txt'* ubicat a la targeta micro-SD de la RPi. S'ha d'afegir un espai al final de la línia de comandes i després afegir la comanda següent:

ip=192.168.137.2

Després d'afegir la comanda, el contingut de l'arxiu *'cmdline.txt'* ha de ser el de la figura IX.III.

```
dwc_otg.lpm_enable=0 console=serial0,115200 console=tty1 root=/dev/mmcblk0p2
rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait dwc_otg.speed=1 ip=192.168.137.2
```

Fig. IX.III - Arxiu de configuració *'cmdline.txt'*.

Guardem els canvis i tornem a posar la targeta micro-SD a la RPi per engegar-la i connectar-nos de forma remota mitjançant una sessió SSH.

Un cop hem accedit a la RPi mitjançant una sessió SSH revisem que el dispositiu tingui connexió a Internet, ja que necessitem descarregar i instal·lar uns paquets addicionals abans d'instal·lar el SW 6LBR.

Al nostre escenari, la RPi accedeix a Internet via Wi-Fi mitjançant un adaptador extern marca LB-LINK i model BL-WN155A. El fitxer de configuració de les interfícies *'interfaces'* ubicat a la ruta *'/etc/network/'* s'ha modificat per tenir connexió a Internet mitjançant l'interfície *wlan* de la RPi, a més, s'ha configurat l'interfície *eth0* per tal que mantingui la IPv4 que se li ha configurat inicialment.

A la figura IX.IV es pot veure el contingut de l'arxiu de configuració *interfaces*.

```
auto lo

iface lo inet loopback

# New configuration for Border Router 6LBR (***)

auto br0

iface br0 inet static
```

```
address 192.168.137.2
netmask 255.255.255.0

pre-up brctl addbr br0
pre-up brctl addif br0 eth0
pre-up ip addr flush dev eth0

post-down ip link set eth0 down
post-down ip link set br0 down
post-down brctl delif br0 eth0
post-down brctl delbr br0


auto tap0

iface tap0 inet dhcp


allow-hotplug wlan0

iface wlan0 inet manual

wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf

iface default inet dhcp
```

Fig. IX.IV - Fitxer de configuració *'interfaces'.*

Després de modificar l'arxiu *'interfaces'* s'han de configurar els paràmetres de la xarxa Wi-Fi que dóna connectivitat a Internet a la RPi. Per fer-ho, modifiquem l'arxiu de configuració *'wpa-supplicant.conf'*, ubicat a la ruta *'/etc/wpa_supplicant/'*. El contingut de l'arxiu *'wpa-supplicant.conf'* és el de la figura IX.V.

```
country=ES
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
        ssid="AndroidAPCubi"
        psk="#nando29"
}
```

Fig. IX.V - Fitxer de configuració *'wpa_supplicant.conf'*

Després de modificar aquests dos arxius de configuració i guardar els canvis, s'ha de reiniciar la RPi perquè els canvis s'apliquin correctament.

Quan ja tenim accés a Internet procedim a instal·lar el SW 6LBR. Descarreguem i instal·lem el paquet addicional *'bridge-utils'* mitjançant la comanda següent:

sudo apt-get install bridge-utils

Aquest paquet es requereix pel mode *'router'* del SW 6LBR que s'utilitza en aquest sistema.

Després d'instal·lar aquest paquet addicional descarreguem i instal·lem el SW 6LBR mitjançant les comandes de la figura IX.VI.

```
sudo wget https://raw.github.com/wiki/cetic/6lbr/releases/cetic-6lbr_1.3.3_armhf.deb
sudo dpkg –i cetic-6lbr_1.3.3_armhf.deb
```

Fig. IX.VI - Comandes per descarregar i instal·lar el SW 6LBR.

Si l'execució de les comandes anteriors no ha retornat cap error reiniciem el servei *6lbr* mitjançant la comanda següent:

sudo service 6lbr restart

L'últim pas és comprovar que el BR està funcionant correctament. Per fer-ho, a l'apartat 4.4 d'aquesta memòria es descriuen els passos a seguir per realitzar aquesta comprovació.

ANNEX X – CODI DELS DRIVERS DELS SENSORS

Drivers del sensor radio:

```
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "lib/sensors.h"
#include "dev/radio-sensor.h"
#include "config.h"

const struct sensors_sensor radio_sensor;

enum {
  ON, OFF
};

static uint8_t state = OFF;

/*---------------------------------------------------------------------------*/

static int value (int type)
{
    return get_lqi();
}
/*---------------------------------------------------------------------------*/

static int status (int type)
{
  switch(type) {
  case SENSORS_ACTIVE:
  case SENSORS_READY:
    return (state == ON);
  }
  return 0;
}
/*---------------------------------------------------------------------------*/

static int configure (int type, int c)
{
  switch(type) {
  case SENSORS_ACTIVE:
    if(c) {
      if(!status(SENSORS_ACTIVE)) {
        state = ON;
      }
      else {
        state = OFF;
      }
    }
  }
  return 0;
}
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR (radio_sensor, RADIO_SENSOR, value, configure, status);
```

Fig. X.I - Arxiu *radio_sensor.c* del driver del sensor radio – lqi.

```
#ifndef __RADIO_SENSOR_H__
#define __RADIO_SENSOR_H__

#include "lib/sensors.h"

extern const struct sensors_sensor radio_sensor;

//extern unsigned int radio_sensor_signal;

static int value (int type);
static int status (int type);
static int configure (int type, int c);

#endif /* __RADIO_SENSOR_H__ */
```

Fig. X.II - Arxiu *radio_sensor.h* del driver del sensor radio – lqi.

Drivers del sensor de nivell de bateria:

```
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "lib/sensors.h"            // To use the sensors_sensor structure.
#include "dev/battery-sensor.h"
#include "config.h"                 // To use the ADC module.
#include "adc.h"                    // To use the ADC module.
#include <time.h>                   // To use the timer module.

const struct sensors_sensor battery_sensor;

enum {
  ON, OFF
};
static uint8_t state = OFF;

/*---------------------------------------------------------------------------*/

static int value (int type)
{
  return adc_vbatt;
}
/*---------------------------------------------------------------------------*/

static int status (int type)
{
  switch(type) {
  case SENSORS_ACTIVE:
  case SENSORS_READY:
    return (state == ON);
  }
  return 0;
}
/*---------------------------------------------------------------------------*/

static int configure (int type, int c)
{
```

```
  switch(type) {
  case SENSORS_ACTIVE:
    if(c) {
      if(!status(SENSORS_ACTIVE)) {

                        //adc_init();
                        state = ON;
      }
    } else {
      state = OFF;
    }
  }
  return 0;
}
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR (battery_sensor, BATTERY_SENSOR, value, configure, status);
```

Fig. X.III - Arxiu *battery_sensor.c* del driver del sensor de nivell de bateria.

```
#ifndef __BATTERY_SENSOR_H__
#define __BATTERY_SENSOR_H__

#include "lib/sensors.h"

extern const struct sensors_sensor battery_sensor;

#define BATTERY_SENSOR  0

static int value (int type);
static int status (int type);
static int configure (int type, int c);

#endif /* __BATTERY-SENSOR_H__ */
```

Fig. X.IV - Arxiu *battery_sensor.h* del driver del sensor de nivell de bateria.

Drivers compartits dels sensors Decagon:

```
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "lib/sensors.h"            // To use the sensors_sensor structure.
#include "dev/decagon_sensors.h"
#include "sys/timer.h"                        // To use the timer module.

uint8_t dades [17] = {0};// To save the data extracted from the reading of UART2 (data from
Decagon digital sensor)
uint8_t raw_EC [5] = {0};           // To save the raw EC (Electroconductivity) value to operate.
uint8_t raw_VWC [5] = {0};          // To save the raw VWC (Volume Water Content) value to
operate.
uint8_t index_EC = 0x00;            // To save the number of chars that have the EC value.
uint8_t index_VWC = 0x00;           // To save the number of chars that have the VWC value.

//-------------------------------------------------------------------------------------------//

void uart2_init (void) {
```

```
        /*Setup the correct registers in order to select the desired functions of every GPIO that
we are using.*/
        GPIO->PAD_PU_SEL.U2RX = 1;          // Pull-ups/pull-downs enabled for
GPIO_19/UART2_RX.
        uart_init(UART2, 1200);             // To read the data from 5TM sensors.
}
//-------------------------------------------------------------------------------------//

uint16_t Checksum (uint8_t * response, uint8_t length) {// To calculate the checksum of the
reading.

        uint8_t i = 0x00;
        uint16_t sum = 0x00;
        uint16_t crc = 0x00;

        for( i = 16; (16-i) < length; i-- ) {     // Adding characters in the response together.
          sum += response[i];
        }

        crc = sum % 64 + 32;     // Convert checksum to a printable character.

        return crc;
}
//-------------------------------------------------------------------------------------//

void read_sensor (int sensnum) {          // To read the chars from the Decagon 5TE or 5TM
sensor.

  uint8_t c = 0x00;                       // To save one character readed from UART2.
  uint8_t d = 0x00;                       // To view the discarded characters.
  uint16_t j = 0x00;                      // Index for raw_data[] uint8_t array.

  uint8_t raw_data [20] = {0};            // Vector to store raw values readed from UART2.
  uint8_t x = 0x00;                               // Index for dades[] uint8_t array.
  uint16_t checksum = 0x00;               // To save the checksum value obtained from
Checksum (...) function.


  static struct timer readtimer;    // To declare a timer to limit the maximum reading sensor time.

  switch(sensnum) {                       // Switch ON the power of the sensor.
        case D5TM1_SENSOR: d5tm1_on();
                           printf("Power ON 5TM sensor 1. \n");
                           break;

        case D5TM2_SENSOR: d5tm2_on();
                           printf("Power ON 5TM sensor 2. \n");
                           break;

        case D5TE_SENSOR: d5te_on();
                           printf("Power ON 5TE sensor. \n");
  }

  timer_set(&readtimer, 3 * CLOCK_SECOND);          // Set the time of the reading timer.

  printf("Waiting data. \n");

  while ( uart2_can_get() ) {             // To discard the previous bad chars and clean the
UART2 SW buffer to read the desired characters.
```

```
        d = uart2_getc();
  }

  while( (c != 0x0A) && (j < 20) ) {                    // Read while the readed char is different of
'0x0A' special char, and the number of readed chars is lower than 20.

        if(uart2_can_get()) {               // Read from UART2 if some information is received.

                c = uart2_getc();

                raw_data[j] = c;
                printf("Raw_data: %02X , j=%d \n", raw_data[j], j);
                j++;
        }

        if (timer_expired(&readtimer)) { break;}  // If read timer is expired quit from the while.
  }


  switch(sensnum) {                         // Switch OFF the power of the sensor.
        case D5TM1_SENSOR: d5tm1_off();
                           printf("Power OFF 5TM sensor 1. \n");
                           break;

        case D5TM2_SENSOR: d5tm2_off();
                           printf("Power OFF 5TM sensor 2. \n");
                           break;

        case D5TE_SENSOR: d5te_off();
                           printf("Power OFF 5TE sensor. \n");
  }

  timer_reset(&readtimer);                  // Reset the readtimer.


  if ((raw_data[j-1] == 0x0A) && (raw_data[j-2] == 0x0D)) {       // The reading might have the
right format.


        while ((raw_data [j-4-x] != 0x20 && (x <= 4))) {   // Extract the Decagon sensor model
and the raw TEMP values.

                dades[16-x] = raw_data [j-4-x];                 // Save the Decagon sensor
model and the raw TEMP values.
                x++;
        }

        dades[16-x] = 0x20;      // Set the 'SPACE' char (0x20) into the right position of the
dades array.

        while ((raw_data [j-5-x] != 0x20 && (x <= 8))) {   // Extract the raw EC value.

                dades[16-x-1] = raw_data [j-5-x];
                raw_EC[3 - index_EC] = raw_data [j-5-x];        // Save the raw EC value.

                x++;
                index_EC++;
        }
```

```
        dades[16-x-1] = 0x20;    // Set the 'SPACE' char (0x20) into the right position of the
dades array.

        while (((raw_data [j-6-x] >= 0x30) && (raw_data [j-6-x] <= 0x39) && (x <= 12))) { //
Extract the raw VWC value.

                dades[16-x-2] = raw_data [j-6-x];
                raw_VWC[3 - index_VWC] = raw_data [j-6-x];    // Save the raw VWC value.

                x++;
                index_VWC++;
        }

        checksum = Checksum (dades, x+2);    // Calculate the checksum to check if the
response has any error.

        if (checksum != (uint16_t) raw_data[j-3]) {          // Check if we have a checksum value
match. Otherwise, discard the reading.

          printf("\n Checksum error! \n");

          memset(&dades[0], 0, sizeof(dades));              // Reset the previous reading and the
related arrays and variables.
          memset(&raw_EC[0], 0, sizeof(raw_EC));
          memset(&raw_VWC[0], 0, sizeof(raw_VWC));
          index_EC = 0x00;
          index_VWC = 0x00;
        }

        c = 0x00;          // Reset the variables (indexes and 'c', used to save every char readed
from UART2).
        j = 0x00;
        x = 0x00;
  }

  else { // The chars readed from UART2 are not good. Are rejected.

        c = 0x00;
        j = 0x00;
  }
}
//-------------------------------------------------------------------------------------//

int get_temp (int sensorid) {      // To obtain the TEMP value from Decagon 5TE or 5TM
sensor.

  int temp, val = 0;               // To save the obtained TEMP value and the previous raw
value.


  read_sensor (sensorid);          // To make the reading of the sensor. Only have to execute
once.


  val = ((dades[12] - 48) * 100) + ((dades[13] - 48) * 10) + (dades[14] - 48);                  //
Convert from uint8_t (char) to int and merge it.

  memset(&dades[0], 0, sizeof(dades));          // Reset the array to save a new reading.

  if ( (val >= 0) && (val <= 920) ) {           // Check if the TEMP raw value is out of range.
```

```
            if (val <= 900) {

              temp = (val - 400);
            }
            else {                    // TEMP decompressed for TEMP raw values that exceed 900.

              val = 5 * (val - 900) + 900;
              temp = (val - 400);
            }
  }
  else { temp = 9999; }             // If a reading is not valid, the result will always be 9999 in all
the magnitudes.

  return temp;   // Tenths of a degree. Instructions of 5TM Integrators Guide (modified because
the value returned by the value() function of sensor data structure of Contiki is a int, not a float.
The division by 10 will do in the CoAP specified resource).
}
//-------------------------------------------------------------------------------------//

int get_vwc (void) {                // To obtain the VWC value from Decagon 5TE or 5TM sensor.

  int vwc = 0;                      // To save the obtained VWC value.


  switch (index_VWC) {              // Convert from uint8_t (char) to int (x - 48) and merge it.

          case 0: vwc = 9999;
                  break;
          case 1: vwc = (raw_VWC[3] - 48);
                  break;
          case 2: vwc = (((raw_VWC[2] - 48) * 10) + (raw_VWC[3] - 48));
                  break;
          case 3: vwc = (((raw_VWC[1] - 48) * 100) + ((raw_VWC[2] - 48) * 10) + (raw_VWC[3] -
48));
                  break;
          case 4: vwc = ((raw_VWC[0] - 48) * 1000) + ((raw_VWC[1] - 48) * 100) + ((raw_VWC[2]
- 48) * 10) + (raw_VWC[3] - 48);
  }

  if ((vwc <= 50) || (vwc >= 4000)) {               // Check if the VWC raw value is out of range.

          vwc = 9999;
  }

  index_VWC = 0x00;                                 // Reset the variables and the arrays to convert
a new value.
  memset(&raw_VWC[0], 0, sizeof(raw_VWC));
  index_EC = 0x00;                                  // Need to reset this variable and this array.
Otherwise, we have an error when make a 5TM
  memset(&raw_EC[0], 0, sizeof(raw_EC));        // petition and next, we make a 5TE petition.

  return vwc;               // The value must be divided by 50 to obtain the raw dielectric output
value. Instructions in Integration's Guide of Decagon 5TM Sensor.
}
//-------------------------------------------------------------------------------------//

int get_ec (void) {                 // To obtain the EC value from Decagon 5TE sensor.

  int ec = 0;                       // To save the obtained EC value.
```

```
  switch (index_EC) {               // Convert from uint8_t (char) to int (x - 48) and merge it.

          case 0: ec = 9999;
                  break;
          case 1: ec = (raw_EC[3] - 48);
                  break;
          case 2: ec = (((raw_EC[2] - 48) * 10) + (raw_EC[3] - 48));
                  break;
          case 3: ec = (((raw_EC[1] - 48) * 100) + ((raw_EC[2] - 48) * 10) + (raw_EC[3] - 48));
                  break;
          case 4: ec = ((raw_EC[0] - 48) * 1000) + ((raw_EC[1] - 48) * 100) + ((raw_EC[2] - 48) *
10) + (raw_EC[3] - 48);
  }


  if ((ec < 0) || (ec > 2300)) {   // Check if the EC raw value is out of range.

          ec = 9999;
  }

  index_EC = 0x00;                                  // Reset the variable and the array to convert a
new EC value.
  memset(&raw_EC[0], 0, sizeof(raw_EC));

  return ec;       // Must be divided by 100. Instructions of 5TE Integrators Guide (modified
because the value returned by the value() function of sensors_sensor structure of Contiki is a
int, not a float. The division by 100 will do in the CoAP specified resource)
}
//-------------------------------------------------------------------------------------//
```

Fig. X.V - Arxiu *decagon_sensors.c* dels drivers dels sensors Decagon.

```
#ifndef __DECAGON_SENSORS_H__
#define __DECAGON_SENSORS_H__

#include "lib/sensors.h"          // To use the sensors_sensor structure.
#include "dev/d5tm1_sensor.h"
#include "dev/d5tm2_sensor.h"
#include "dev/d5te_sensor.h"

#define D5TM1_SENSOR 0
#define D5TM2_SENSOR 1
#define D5TE_SENSOR 2

void uart2_init (void);
uint16_t Checksum (uint8_t * response, uint8_t length);
void read_sensor (int sensnum);
int get_temp (int sensorid);
int get_vwc (void);
int get_ec (void);

#endif /* __DECAGON_SENSORS_H__ */
```

Fig. X.VI - Arxiu *decagon_sensors.h* dels drivers dels sensors Decagon.

Driver del sensor Decagon 5TE:

```
#include <stdlib.h>        // To use the printf() and sprintf() functions.
#include <string.h>
#include "contiki.h"
#include "lib/sensors.h"            // To use the sensors_sensor structure.
#include "dev/decagon_sensors.h"
#include "dev/d5te_sensor.h"
#include "sys/timer.h"              // To use the timer module.


// Check the MC1322X Reference Manual to see what register must to modify to output a '1' by
the GPIO. In our case, the GPIO_PAD_DIR0 32 bits register.

#define PWR_ON1 0x1000000           // To put '1' or '0' in the configured GPIO.
#define PWR_OFF1 0xFEFFFFFF         // ~ON.

const struct sensors_sensor d5te_sensor;

enum {
  ON, OFF
};
static uint8_t state = OFF;

//-------------------------------------------------------------------------------------//

void d5te_init (void) {

        /* Setup the correct registers in order to select the desired functions of every GPIO that
we are using. */
        *GPIO_FUNC_SEL1 |= 0x30000;    // Select the function of GPIO_24/KBI_2 (func. 3 -
alternate).
        *GPIO_PAD_DIR0 |= 0x1000000;   // Setting as output the GPIO_24/KBI_2.
}
//-------------------------------------------------------------------------------------//

void d5te_on (void) {

   *GPIO_DATA0 |= PWR_ON1;// Put a "1" in the GPIO_24/KBI_2.

}
//-------------------------------------------------------------------------------------//

void d5te_off (void) {

   static struct timer offtimer_5te;                    // Declare a timer.

   timer_set(&offtimer_5te, CLOCK_SECOND);              // Set timer.

   *GPIO_DATA0 &= PWR_OFF1;         // Put a "0" in the GPIO_24/KBI_2.

   while (timer_expired(&offtimer_5te)) {break; }       // Wait until timer expires and exit from
while loop.

   timer_reset(&offtimer_5te);              // Reset the timer.
}
//-------------------------------------------------------------------------------------//
```

```
static int value (int type)
{
  switch(type) {

  case D5TE_SENSOR_TEMP:
    return get_temp(2);    // 0 - Sensor 1, 1 - Sensor 2, 2 - Sensor 3.

  case D5TE_SENSOR_VWC:
    return get_vwc();

  case D5TE_SENSOR_EC:
    return get_ec();
  }
  return 0;
}
//---------------------------------------------------------------------------------//

static int status (int type)
{
  switch(type) {
  case SENSORS_ACTIVE:
  case SENSORS_READY:
    return (state == ON);
  }
  return 0;
}
//---------------------------------------------------------------------------------//

static int configure (int type, int c)
{
  switch(type) {
  case SENSORS_ACTIVE:
    if(c) {
      if(!status(SENSORS_ACTIVE)) {

            uart2_init();       // To setup and init UART2. Must only be called once.
            d5te_init();

        state = ON;
      }
    } else {
            d5te_off();
            state = OFF;
    }
  }
  return 0;
}
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR (d5te_sensor, "d5te", value, configure, status);
```

Fig. X.VII - Arxiu *d5te_sensor.c* del driver del sensor Decagon 5TE.

```
#ifndef __D5TE_SENSOR_H__
#define __D5TE_SENSOR_H__

#include "lib/sensors.h"     // To use the sensors_sensor structure.

#define D5TE_SENSOR_TEMP  0
#define D5TE_SENSOR_VWC   1                  // Volume Water Content.
```

```
#define D5TE_SENSOR_EC      2                   // Electroconductivity.

extern const struct sensors_sensor d5te_sensor;

void d5te_init (void);
void d5te_on (void);
void d5te_off (void);

static int value (int type);
static int status (int type);
static int configure (int type, int c);

#endif /* __D5TE_SENSOR_H__ */
```

Fig. X.VIII - Arxiu *d5te_sensor.h* del driver del sensor Decagon 5TE.

Driver del sensor 5TM 1:

```
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "lib/sensors.h"            // To use the sensors_sensor structure.
#include "dev/decagon_sensors.h"
#include "dev/d5tm1_sensor.h"
#include "sys/timer.h"                      // To use the timer module.

// Check the MC1322X Reference Manual to see what register must to modify to output '1' by
the GPIO. In our case the GPIO_PAD_DIR0 32 bits register.


#define PWR_ON1 0x10000000          // To put '1' or '0' in the configured GPIO.
#define PWR_OFF1 0xEFFFFFFF         // ~ON.

const struct sensors_sensor d5tm1_sensor;

enum {
  ON, OFF
};
static uint8_t state = OFF;

//-------------------------------------------------------------------------------------------//
void d5tm1_init (void) {

        /* Setup the correct registers in order to select the desired functions of every GPIO that
we are using. */

        *GPIO_FUNC_SEL1 |= 0x3000000;       // Select the function of GPIO_28/KBI_6 (func.
3 - alternate).
        *GPIO_PAD_DIR0 |= 0x10000000;       // Setting as output the GPIO_28/KBI_6.
}

//-------------------------------------------------------------------------------------------//
void d5tm1_on (void) {

   *GPIO_DATA0 |= PWR_ON1;          // Put a "1" in the GPIO_28/KBI_6.
}
//-------------------------------------------------------------------------------------------//
void d5tm1_off (void) {
```

```
  static struct timer offtimer1;                // Declare a timer.

  timer_set(&offtimer1, CLOCK_SECOND);      // Set the timer.

  *GPIO_DATA0 &= PWR_OFF1;          // Put a "0" in the GPIO_28/KBI_6.

  while (timer_expired(&offtimer1)) {break; }  // Wait until timer expires and exit from while.

  timer_reset(&offtimer1);          // Reset the timer.
}

/*---------------------------------------------------------------------------*/
static int value (int type)
{
 switch(type) {

 case D5TM1_SENSOR_TEMP:
   return get_temp(0);   // 0 - Sensor 1, 1 - Sensor 2, 2 - Sensor 3.

 case D5TM1_SENSOR_VWC:
   return get_vwc();
 }

 return 0;
}
/*---------------------------------------------------------------------------*/
static int status (int type)
{
 switch(type) {
 case SENSORS_ACTIVE:
 case SENSORS_READY:
   return (state == ON);
 }
 return 0;
}
/*---------------------------------------------------------------------------*/
static int configure (int type, int c)
{
 switch(type) {
 case SENSORS_ACTIVE:
   if(c) {
     if(!status(SENSORS_ACTIVE)) {

                    d5tm1_init();
       state = ON;
     }
   } else {
                    d5tm1_off();
                    state = OFF;
   }
 }
 return 0;
}
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR (d5tm1_sensor, "d5tm1", value, configure, status);
```

Fig. X.IX - Arxiu *d5tm1_sensor.c* del driver del sensor Decagon 5TM 1.

```
#ifndef __D5TM1_SENSOR_H__
#define __D5TM1_SENSOR_H__

#include "lib/sensors.h"

#define D5TM1_SENSOR_TEMP 0
#define D5TM1_SENSOR_VWC  1  // Volume Water Content.

extern const struct sensors_sensor d5tm1_sensor;

void d5tm1_init (void);
void d5tm1_on (void);
void d5tm1_off (void);

static int value (int type);
static int status (int type);
static int configure (int type, int c);

#endif /* __D5TM1_SENSOR_H__ */
```

Fig. X.X - Arxiu *d5tm1_sensor.h* del driver del sensor Decagon 5TM 1.

Driver del sensor Decagon 5TM 2:

```
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "lib/sensors.h"              // To use the sensors_sensor structure.
#include "dev/decagon_sensors.h"
#include "dev/d5tm2_sensor.h"

#include "sys/timer.h"

#define PWR_ON2 0x20000000            // To put '1' or '0' in the configured GPIO.
#define PWR_OFF2 0xDFFFFFFF           // ~ON.

const struct sensors_sensor d5tm2_sensor;

enum {
  ON, OFF
};
static uint8_t state = OFF;

//------------------------------------------------------------------------------//
void d5tm2_init (void) {

   /*Setup the correct registers in order to select the desired functions of every GPIO that we
are using.*/
   *GPIO_FUNC_SEL1 |= 0xC000000;   // Select the function of GPIO_29/KBI_7 (func. 3 -
alternate).
   *GPIO_PAD_DIR0 |= 0x20000000;   // Setting as output the GPIO_29/KBI_7.
}
//------------------------------------------------------------------------------//
void d5tm2_on (void) {

   *GPIO_DATA0 |= PWR_ON2;   // Put a "1" in the GPIO_29/KBI_7.
}
//------------------------------------------------------------------------------//
```

```
void d5tm2_off (void) {

  static struct timer offtimer2;                // Declare a timer.

  timer_set(&offtimer2, CLOCK_SECOND);       // Set the timer.

  *GPIO_DATA0 &= PWR_OFF2;          // Put a "0" in the GPIO_29/KBI_7.

  while (timer_expired(&offtimer2)) {break; }   // Wait until timer expires and exit from while.

  timer_reset(&offtimer2);          // Reset the timer.
}
/*---------------------------------------------------------------------------*/
static int value (int type)
{
 switch(type) {

 case D5TM2_SENSOR_TEMP:
   return get_temp(1);   // 0 - Sensor 1, 1 - Sensor 2, 2 - Sensor 3.

 case D5TM2_SENSOR_VWC:
   return get_vwc();
 }
 return 0;
}
/*---------------------------------------------------------------------------*/
static int status (int type)
{
 switch(type) {
 case SENSORS_ACTIVE:
 case SENSORS_READY:
   return (state == ON);
 }
 return 0;
}
/*---------------------------------------------------------------------------*/
static int configure (int type, int c)
{
 switch(type) {
 case SENSORS_ACTIVE:
   if(c) {
     if(!status(SENSORS_ACTIVE)) {
          d5tm2_init();
          state = ON;
     }
   } else {
           d5tm2_off();
           state = OFF;
   }
 }
 return 0;
}
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR (d5tm2_sensor, "d5tm2", value, configure, status);
```

Fig. X.XI - Arxiu *d5tm2_sensor.c* del driver del sensor Decagon 5TM 2.

```
#ifndef __D5TM2_SENSOR_H__
#define __D5TM2_SENSOR_H__

#include "lib/sensors.h"


#define D5TM2_SENSOR_TEMP 0
#define D5TM2_SENSOR_VWC  1  // Volume Water Content.

extern const struct sensors_sensor d5tm2_sensor;

void d5tm2_init (void);
void d5tm2_on (void);
void d5tm2_off (void);

static int value (int type);
static int status (int type);
static int configure (int type, int c);

#endif /* __D5TM2_SENSOR_H__ */
```

Fig. X.XII - Arxiu *d5tm2_sensor.h* del driver del sensor Decagon 5TM 2.

ANNEX XI – CODI DEL SERVIDOR COAP

```
/* Servidor CoAP seguint la implementació de /home/user/contiki/examples/er-rest-example/er-example-server.c
 * S'ha modificat l'arxiu afegint les línies de codi necessàries per crear els següents recursos:
 *  - Radio (mostra el valor en dBm de l'indicador "LQI" requerit per l'estàndard IEEE 802.15.4)
 *  - Battery (recull la mostra presa per l'ADC mitjançant el pin dedicat).
 *  - D5TE (sensor digital Decagon 5TE de temp, ec i vwc).
 *  - D5TM1 (sensor digital Decagon 5TM de temp i vwc).
 *  - D5TM2 (sensor digital Decagon 5TM de temp i vwc).
 *  - PT100 (Sensor analògic RTD).
 * Els recursos aconsegueixen les dades seguint l'estructura de sensors de Contiki
 * (arxius .c i .h a la carpeta ~/contiki/platform/econotag/dev/).
 */

                        /* Declare the necessary libraries */
//-------------------------------------------------------------------------------------------//
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "contiki-net.h"
#include "erbium.h"
//-------------------------------------------------------------------------------------------//

                        /* Activate the desired resources */
//-------------------------------------------------------------------------------------------//
#define REST_RES_BATTERY 1
#define REST_RES_RADIO 1
#define REST_RES_D5TMSENSORS 1
#define REST_RES_D5TESENSOR 1
#define REST_RES_PT100SENSOR 1
//-------------------------------------------------------------------------------------------//

            /* Declare the necessary libraries to work with the sensors */
//-------------------------------------------------------------------------------------------//
#if defined (PLATFORM_HAS_BATTERY)
#include "dev/battery-sensor.h"
#endif
#if defined (PLATFORM_HAS_RADIO)
#include "dev/radio-sensor.h"
#endif
#if defined (PLATFORM_HAS_D5TMSENSOR)
#include "dev/decagon_sensors.h"
#endif
#if defined (PLATFORM_HAS_PT100SENSOR)
#include "dev/pt100_sensor.h"
#endif
//-------------------------------------------------------------------------------------------//


#define DEBUG 1      // To print the debug information (IPv6 addr, readings from the sensors, etc.)


/* For CoAP-specific example: not required for normal RESTful Web service. */
#if WITH_COAP == 13
#include "er-coap-13.h"
```

```
#else
#warning "Erbium example without CoAP-specifc functionality"
#endif /* CoAP-specific example */

#if DEBUG
#define PRINTF(...) printf(__VA_ARGS__)
#define PRINT6ADDR(addr)
PRINTF("[%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x]",
((uint8_t *)addr)[0], ((uint8_t *)addr)[1], ((uint8_t *)addr)[2], ((uint8_t *)addr)[3], ((uint8_t *)addr)[4], ((uint8_t
*)addr)[5], ((uint8_t *)addr)[6], ((uint8_t *)addr)[7], ((uint8_t *)addr)[8], ((uint8_t *)addr)[9], ((uint8_t
*)addr)[10], ((uint8_t *)addr)[11], ((uint8_t *)addr)[12], ((uint8_t *)addr)[13], ((uint8_t *)addr)[14], ((uint8_t
*)addr)[15])
#define PRINTLLADDR(lladdr) PRINTF("[%02x:%02x:%02x:%02x:%02x:%02x]",(lladdr)->addr[0], (lladdr)-
>addr[1], (lladdr)->addr[2], (lladdr)->addr[3],(lladdr)->addr[4], (lladdr)->addr[5])
#else
#define PRINTF(...)
#define PRINT6ADDR(addr)
#define PRINTLLADDR(addr)
#endif


uint32_t msgcount = 0;     // To count the messages sent.


                                  /* Battery Level Resource */
/*---------------------------------------------------------------------------*/
#if REST_RES_BATTERY && defined (PLATFORM_HAS_BATTERY)
RESOURCE(battery, METHOD_GET, "sensors/Battery", "title=\"Battery status\";rt=\"Battery\"");

void battery_handler (void* request, void* response, uint8_t *buffer, uint16_t preferred_size, int32_t *offset)
{

  int battery = battery_sensor.value(0);   // Make the reading of the battery level.

  msgcount++;     // Increment the message counter.

  /*  Get the header of the petition message received and response accordingly.  */
  const uint16_t *accept = NULL;
  int num = REST.get_header_accept(request, &accept);

  if ((num==0) || (num && accept[0]==REST.type.TEXT_PLAIN)) // If the format of the required data is
different of TEXT_PLAIN refuse the petition.
  {
    REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
    snprintf((char *)buffer, REST_MAX_CHUNK_SIZE, "%d::%04u mV", msgcount, battery);        // Format
the data and save it to the output buffer.

    REST.set_response_payload(response, buffer, strlen((char *)buffer));
  }
  else  {

    REST.set_response_status(response, REST.status.NOT_ACCEPTABLE);
    const char *msg = "Supporting content-types text/plain";
    REST.set_response_payload(response, msg, strlen(msg));
  }

}
#endif /* PLATFORM_HAS_BATTERY && REST_RES_BATTERY */

                                  /* Radio sensor Resource */
```

```
/*---------------------------------------------------------------------------*/
#if REST_RES_RADIO && defined (PLATFORM_HAS_RADIO)
RESOURCE(radio, METHOD_GET, "sensors/Radio", "title=\"Radio LQI\";rt=\"Radio LQI Sensor\"");

void
radio_handler(void* request, void* response, uint8_t *buffer, uint16_t preferred_size, int32_t *offset)
{

  int lqi = radio_sensor.value(0);   // Make the reading of the LQI (Link Quality Indicator) value.

  lqi = ((lqi / 3) - 100);                // Following the advices in MC1322x Reference Manual page 169.

  msgcount++;     // Increment the message counter.

  /*  Get the header of the petition message received and response accordingly.  */
  const uint16_t *accept = NULL;
  int num = REST.get_header_accept(request, &accept);

  if ((num==0) || (num && accept[0]==REST.type.TEXT_PLAIN)) // If the format of the required data is
different of TEXT_PLAIN refuse the petition.
  {
    REST.set_header_content_type(response, REST.type.TEXT_PLAIN);

    snprintf((char *)buffer, REST_MAX_CHUNK_SIZE, "%d::%d dBm", msgcount, lqi);     // Format the data
and save it to the output buffer.

    REST.set_response_payload(response, buffer, strlen((char *)buffer));
  }
  else  {

    REST.set_response_status(response, REST.status.NOT_ACCEPTABLE);
    const char *msg = "Supporting content-types text/plain";
    REST.set_response_payload(response, msg, strlen(msg));
  }

}
#endif /* PLATFORM_HAS_RADIO && REST_RES_RADIO */

                                  /* PT100 analog sensor Resource */
/*---------------------------------------------------------------------------*/
#if REST_RES_PT100SENSOR && defined (PLATFORM_HAS_PT100SENSOR)
RESOURCE(pt100, METHOD_GET, "sensors/PT100", "title=\"Analog temp\";rt=\"PT100 Sensor\"");

void pt100_handler (void* request, void* response, uint8_t *buffer, uint16_t preferred_size, int32_t *offset)
{

  int pt100 = pt100_sensor.value(0);       // Make the reading of the PT100 analog sensor.

  float analogtemp, ft, m, b = 0.0;           // To save the final temp value and the callibration curves values
(slopes and independent terms).

  PRINTF("pt100_ADC = %d\n", pt100); // Print if DEBUG is set to 1.

  if (pt100 == 0) {           // If the reading is equal to 0 make another reading.

          printf("pt100_ADC = 0. Make another reading \n");
          pt100 = pt100_sensor.value(0);
  }

  PRINTF("pt100_ADC = %d\n", pt100); // Print if DEBUG is set to 1.
```

```
  // Set the slope and the independent term of the callibration curve.
  if ( (pt100 >= 0) && (pt100 <= 1587) ) {// ADC = 1587 -> T = -0.6 ºC

          m = 0.034883721;
          b = -55.96046512;
  }
  else if ( (pt100 > 1587) && (pt100 <= 1942) ) {  // ADC = 1942 -> T = 9.6 ºC

          m = 0.028527607;
          b = -45.8006135;
  }
  else if ( (pt100 > 1942) && (pt100 <= 2263) ) {  // ADC = 2263 -> T = 18.8 ºC

          m = 0.03017656501;
          b = -49.09919743;
  }

  else if ( (pt100 > 2263) && (pt100 < 2978) ) {    // ADC = 2978 -> T = 41.6 ºC

          m = 0.03192771084;
          b = -53.84457831;
  }

  else {

          m = 0.035443038;
          b = -64.1;
  }

  PRINTF("pt100_ADC = %d\n", pt100); // Print if DEBUG is set to 1.

  if (pt100 == 0) {           // If the pt100 readed value is equal to 0, discard the reading. Else, calculate the
temperature from the ADC reading.

          analogtemp = 9999;
  }
  else {
          analogtemp = (((float) pt100) * m) + b;
  }

  msgcount++;     // Increment the message counter.

  /*  Get the header of the petition message received and response accordingly.  */
  const uint16_t *accept = NULL;
  int num = REST.get_header_accept(request, &accept);

  if ((num==0) || (num && accept[0]==REST.type.TEXT_PLAIN))// If the format of the required data is
different of TEXT_PLAIN refuse the petition.
  {
    REST.set_header_content_type(response, REST.type.TEXT_PLAIN);

    if (analogtemp >= 0) {            // Format the data and save it to the output buffer.

          ft = (float) ((int) analogtemp);
          snprintf((char *)buffer, REST_MAX_CHUNK_SIZE, "%d::Ta %d.%01d", msgcount, (int)
analogtemp, (int) ((analogtemp - ft) * 10));
    }
    else {
          analogtemp = -analogtemp;
```

```
          ft = (float) ((int) analogtemp);
          snprintf((char *)buffer, REST_MAX_CHUNK_SIZE, "%d::Ta -%d.%01d", msgcount, (int)
analogtemp, (int) ((analogtemp - ft) * 10));
    }

    REST.set_response_payload(response, (uint8_t *)buffer, strlen((char *)buffer));
  }
  else  {

    REST.set_response_status(response, REST.status.NOT_ACCEPTABLE);
    const char *msg = "Supporting content-types text/plain";
    REST.set_response_payload(response, msg, strlen(msg));
  }

}
#endif /* PLATFORM_HAS_PT100SENSOR && REST_RES_PT100SENSOR */


                              /* Decagon 5TE sensor Resource */
/*---------------------------------------------------------------------------------------------*/
#if REST_RES_D5TESENSOR && defined (PLATFORM_HAS_D5TESENSOR)
RESOURCE(d5te_sensor, METHOD_GET, "sensors/D5TE", "title=\"Temp, VWC & EC\";rt=\"D5TE
Sensor\"");

void d5te_sensor_handler (void* request, void* response, uint8_t *buffer, uint16_t preferred_size, int32_t
*offset)
{

  int temp_i,vwc_i, ec_i = 0;       // To save the values readed by the driver.
  float per, vwc_f, fv = 0.0;       // To operate with the previous raw values.
  char temp [8] = {0};              // To save the formatted TEMP, VWC and EC values to send.
  char vwc [11] = {0};              // Formats:
  char ec [9] = {0};                // For TEMP -> T -XX.X , for VWC -> VWC X.XXX and for EC -> EC XX.XX


// To get the sensor values (TEMP, EC and VWC) make the petitions in this order.
  temp_i = d5te_sensor.value(D5TE_SENSOR_TEMP);   // (TEMP, EC and VWC), because in the Decagon
sensors driver, the data readed from
  ec_i = d5te_sensor.value(D5TE_SENSOR_EC);          // the digital sensor are made from the temp
function. And also, due to both sensors
  vwc_i = d5te_sensor.value(D5TE_SENSOR_VWC);     // (5TE and 5TM) are sharing a part of the driver, if
we make the VWC petition first,                     // the EC value will always be 9999.

  PRINTF("temp_i = %d\n", temp_i);       // Print if DEBUG is set to 1.
  PRINTF("ec_i = %d\n", ec_i);
  PRINTF("vwc_i = %d\n", vwc_i);

  if (temp_i != 9999) {     // Do while the "temp_i" value is not out of range.

          if ( temp_i >= 0) {          // Format the data.

             sprintf(temp, "T %d.%d", (temp_i/10), (temp_i%10));
          }

    else {
          temp_i = -temp_i;
          sprintf(temp, "T -%d.%d", (temp_i/10), (temp_i%10));
    }

  }
```

```
  else { sprintf(temp, "T %d", temp_i); }


  if (ec_i != 9999) {         // Do while the "ec_i" value is not out of range.

          if (ec_i > 10) {    // Format the data.

            printf(ec, "EC %d.%02d", (ec_i/100), (ec_i%100));
          }
          else {
            sprintf(ec, "EC %d.0%d", (ec_i/100), (ec_i%10));
          }
  }
  else { sprintf(ec, "EC %d", ec_i); }


  if (vwc_i != 9999) {       // Do while the "vwc_i" value is not out of range.

          per = vwc_i / 50.0;

          vwc_f = ( 4.3e-6 * (per*per*per) ) - ( 5.5e-4 * (per*per) ) + ( 2.92e-2 * per ) -5.3e-2;     // Topp
Equation. Followed by the Decagon 5TE Integrators Guide.

          if ( vwc_f >= 0) {            // Format the data.

            fv = (float) ((int) vwc_f);
            sprintf(vwc, "VWC %d.%03d", (int) vwc_f, (int) ((vwc_f - fv) * 1000));
          }

          else {

            vwc_f = -vwc_f;
            fv =  (float) ((int) vwc_f);
            sprintf(vwc, "VWC -%d.%03d", (int) vwc_f , (int) ((vwc_f - fv) * 1000));
          }
  }
  else { sprintf(vwc, "%d", vwc_i); };

  msgcount++;     // Increment the message counter.

  /*  Get the header of the petition message received and response accordingly.  */
  const uint16_t *accept = NULL;
  int num = REST.get_header_accept(request, &accept);

  if ((num==0) || (num && accept[0]==REST.type.TEXT_PLAIN)) // If the format of the required data is
different of TEXT_PLAIN refuse the petition.
  {
    REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
    snprintf((char *)buffer, REST_MAX_CHUNK_SIZE, "%d::%s::%s::%s", msgcount, temp, ec, vwc);       //
Save the data to the output buffer.

    REST.set_response_payload(response, (uint8_t *)buffer, strlen((char *)buffer));
  }
  else
  {
    REST.set_response_status(response, REST.status.NOT_ACCEPTABLE);
    const char *msg = "Supporting content-types text/plain";
    REST.set_response_payload(response, msg, strlen(msg));
  }
```

```
}
#endif /* PLATFORM_HAS_D5TESENSOR && REST_RES_D5TESENSOR */


                                /* Decagon 5TM 1 sensor Resource */
/*---------------------------------------------------------------------------*/
#if REST_RES_D5TMSENSORS && defined (PLATFORM_HAS_D5TMSENSOR)
RESOURCE(d5tm1_sensor, METHOD_GET, "sensors/D5TM1", "title=\"Temp & VWC \";rt=\"D5TM1
Sensor\"");

void d5tm1_sensor_handler (void* request, void* response, uint8_t *buffer, uint16_t preferred_size, int32_t
*offset)
{

  int temp_i,vwc_i = 0;              // To save the values readed by the driver.
  float per, vwc_f, fv = 0.0;        // To operate with the previous raw values
  char temp [8] = {0};               // To save the formatted TEMP and VWC to send.
  char vwc [11] = {0};               // Formats:
                                     // For TEMP -> T -XX.X , for VWC -> VWC X.XXX


  temp_i = d5tm1_sensor.value(D5TM1_SENSOR_TEMP);        // To get the sensor values (Temp and
VWC). Make the temp petition first, because in the Decagon sensors driver,
  vwc_i = d5tm1_sensor.value(D5TM1_SENSOR_VWC);          // the data readed from the digital sensor
are made from temp function.

  PRINTF("temp_i = %d\n", temp_i);       // Print if DEBUG is set to 1.
  PRINTF("vwc_i = %d\n", vwc_i);

  if (temp_i != 9999) {   // Do while the "temp_i" value is not out of range.

          if ( temp_i >= 0) {          // Format the data.

             sprintf(temp, "T %d.%d", (temp_i/10), (temp_i%10));
           }

           else {
             temp_i = -temp_i;
             sprintf(temp, "T -%d.%d", (temp_i/10), (temp_i%10));
           }

  }
  else {sprintf(temp, "T %d", temp_i);}


  if (vwc_i != 9999) {   // Do while the "vwc_i" value is not out of range.

     per = vwc_i / 50.0;

     vwc_f = ( 4.3e-6 * (per*per*per) ) - ( 5.5e-4 * (per*per) ) + ( 2.92e-2 * per ) -5.3e-2;     // Topp Equation.
Followed by the Decagon 5TE Integrators Guide.

             if ( vwc_f >= 0) {          // Format the data.

                fv = (float) ((int) vwc_f);
                sprintf(vwc, "VWC %d.%03d", (int) vwc_f, (int) ((vwc_f - fv) * 1000));
             }
             else {
                vwc_f = -vwc_f;
                fv =  (float) ((int) vwc_f);
```

```
                sprintf(vwc, "VWC -%d.%03d", (int) vwc_f , (int) ((vwc_f - fv) * 1000));
            }
  }
  else { sprintf(vwc, "VWC %d", vwc_i); };

  msgcount++;     // Increment the message counter.

  /*  Get the header of the petition message received and response accordingly.  */
  const uint16_t *accept = NULL;
  int num = REST.get_header_accept(request, &accept);

  if ((num==0) || (num && accept[0]==REST.type.TEXT_PLAIN)) // If the format of the required data is
different of TEXT_PLAIN refuse the petition.
  {
    REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
    snprintf((char *)buffer, REST_MAX_CHUNK_SIZE, "%d::%s::%s", msgcount, temp, vwc);       // Save the
data to the output buffer.

    REST.set_response_payload(response, (uint8_t *)buffer, strlen((char *)buffer));
  }
  else
  {
    REST.set_response_status(response, REST.status.NOT_ACCEPTABLE);
    const char *msg = "Supporting content-types text/plain";
    REST.set_response_payload(response, msg, strlen(msg));
  }

}
#endif /* PLATFORM_HAS_D5TMSENSOR && REST_RES_D5TMSENSORS */


                              /* Decagon 5TM 2 sensor Resource */
/*****************************************************************************************/
#if REST_RES_D5TMSENSORS && defined (PLATFORM_HAS_D5TMSENSOR)
RESOURCE(d5tm2_sensor, METHOD_GET, "sensors/D5TM2", "title=\"Temp & VWC \";rt=\"D5TM2
Sensor\"");

void d5tm2_sensor_handler (void* request, void* response, uint8_t *buffer, uint16_t preferred_size, int32_t
*offset)
{

  int temp_i,vwc_i = 0;             // To save the values readed by the driver.
  float per, vwc_f, fv = 0.0;       // To operate with the previous raw values
  char temp [8] = {0};              // To save the formatted TEMP and VWC to send.
  char vwc [11] = {0};              // Formats:
                                    // For TEMP -> T -XX.X , for VWC -> VWC X.XXX


  temp_i = d5tm2_sensor.value(D5TM1_SENSOR_TEMP);       // To get the sensor values (Temp and
VWC). Make the temp petition first, because in the Decagon sensors driver,
  vwc_i = d5tm2_sensor.value(D5TM1_SENSOR_VWC);         // the data readed from the digital sensor
are made from temp function.

  PRINTF("temp_i = %d\n", temp_i);      // Print if DEBUG is set to 1.
  PRINTF("vwc_i = %d\n", vwc_i);

  if (temp_i != 9999) {   // Do while the "temp_i" value is not out of range.

          if ( temp_i >= 0) {          // Format the data.
```

```
            sprintf(temp, "T %d.%d", (temp_i/10), (temp_i%10));
          }

      else {
          temp_i = -temp_i;
          sprintf(temp, "T -%d.%d", (temp_i/10), (temp_i%10));
      }

 }
 else {sprintf(temp, "T %d", temp_i);}


 if (vwc_i != 9999) {   // Do while the "vwc_i" value is not out of range.

            per = vwc_i / 50.0;

            vwc_f = ( 4.3e-6 * (per*per*per) ) - ( 5.5e-4 * (per*per) ) + ( 2.92e-2 * per ) -5.3e-2;     // Topp
Equation. Followed by the Decagon 5TE Integrators Guide.

            if ( vwc_f >= 0) {          // Format the data.

               fv = (float) ((int) vwc_f);
               sprintf(vwc, "VWC %d.%03d", (int) vwc_f, (int) ((vwc_f - fv) * 1000));
            }
            else {
               vwc_f = -vwc_f;
               fv =  (float) ((int) vwc_f);
               sprintf(vwc, "VWC -%d.%03d", (int) vwc_f , (int) ((vwc_f - fv) * 1000));
            }
 }
 else { sprintf(vwc, "VWC %d", vwc_i); };

 msgcount++;     // Increment the message counter.

 /*  Get the header of the petition message received and response accordingly.  */
 const uint16_t *accept = NULL;
 int num = REST.get_header_accept(request, &accept);

 if ((num==0) || (num && accept[0]==REST.type.TEXT_PLAIN)) // If the format of the required data is
different of TEXT_PLAIN refuse the petition.
 {
   REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
   snprintf((char *)buffer, REST_MAX_CHUNK_SIZE, "%d::%s::%s", msgcount, temp, vwc);       // Save the
data to the output buffer.

   REST.set_response_payload(response, (uint8_t *)buffer, strlen((char *)buffer));
 }
 else  {

   REST.set_response_status(response, REST.status.NOT_ACCEPTABLE);
   const char *msg = "Supporting content-types text/plain";
   REST.set_response_payload(response, msg, strlen(msg));
 }

}
#endif /* PLATFORM_HAS_D5TMSENSOR && REST_RES_D5TMSENSORS */

                              /* Start the CoAP server process */
/*---------------------------------------------------------------------------------------------*/
PROCESS(rest_server, "Erbium Server");
```

```
AUTOSTART_PROCESSES(&rest_server);

PROCESS_THREAD(rest_server, ev, data)
{
  PROCESS_BEGIN();  // Start the REST Server process.

  PRINTF("Starting Erbium Server\n");

#ifdef RF_CHANNEL
  PRINTF("RF channel: %u\n", RF_CHANNEL);
#endif
#ifdef IEEE802154_PANID
  PRINTF("PAN ID: 0x%04X\n", IEEE802154_PANID);
#endif

  PRINTF("uIP buffer: %u\n", UIP_BUFSIZE);
  PRINTF("LL header: %u\n", UIP_LLH_LEN);
  PRINTF("IP+UDP header: %u\n", UIP_IPUDPH_LEN);
  PRINTF("REST max chunk: %u\n", REST_MAX_CHUNK_SIZE);

  /* Initialize the REST engine. */
  rest_init_engine();

          /* Initialization of the desired resources and of the corresponding sensors */
//-------------------------------------------------------------------------------------//
#if defined (PLATFORM_HAS_BATTERY) && REST_RES_BATTERY
  SENSORS_ACTIVATE(battery_sensor);
  rest_activate_resource(&resource_battery);
  PRINTF("Battery Sensor started\n");
#endif
#if defined (PLATFORM_HAS_RADIO) && REST_RES_RADIO
  SENSORS_ACTIVATE(radio_sensor);
  rest_activate_resource(&resource_radio);
  PRINTF("Radio Sensor started\n");
#endif
#if defined (PLATFORM_HAS_D5TMSENSOR) && REST_RES_D5TMSENSORS
  SENSORS_ACTIVATE(d5tm1_sensor);
  SENSORS_ACTIVATE(d5tm2_sensor);
  rest_activate_resource(&resource_d5tm1_sensor);
  rest_activate_resource(&resource_d5tm2_sensor);
  PRINTF("D5TM Sensors started\n");
#endif
#if defined (PLATFORM_HAS_D5TESENSOR) && REST_RES_D5TESENSOR
  SENSORS_ACTIVATE(d5te_sensor);
  rest_activate_resource(&resource_d5te_sensor);
  PRINTF("D5TE Sensor started\n");
#endif
#if defined (PLATFORM_HAS_PT100SENSOR) && REST_RES_PT100SENSOR
  SENSORS_ACTIVATE(pt100_sensor);
  rest_activate_resource(&resource_pt100);
  PRINTF("PT100 Sensor started\n");
#endif
//-------------------------------------------------------------------------------------//

  PROCESS_END();
}
```

Fig. XI.I – Arxiu *coap-server.c* que conté el codi en llenguatge *C* del servidor CoAP.

ANNEX XII – SCRIPTS D'AUTOMATITZACIÓ DE PROCESSOS

Script per automatitzar les peticions als recursos CoAP des del BR:

```
#! /bin/bash

#Execute the python scripts to get the desired data from sensors and save the results

sudo python /home/pi/CoAP/get_coap_5tm.py >> /home/pi/CoAP/get_coap_5tm.log &
PID=$!
sleep 15
sudo kill $PID

sudo python /home/pi/CoAP/get_coap_pt100.py >> /home/pi/CoAP/get_coap_pt100.log &
PID1=$!
sleep 15
sudo kill $PID1

sudo python /home/pi/CoAP/get_coap_lqi.py >> /home/pi/CoAP/get_coap_lqi.log &
PID2=$!
sleep 8
sudo kill $PID2

sudo python /home/pi/CoAP/get_coap_batt.py >> /home/pi/CoAP/get_coap_batt.log &
PID3=$!
sleep 8
sudo kill $PID3
```

Fig. XII.I Arxiu *coap_petitions.sh*.

L'script de la figura XII.I s'ha d'executar mitjançant el dimoni *cron* per automatitzar les peticions als recursos CoAP. Aquest script de comandes guarda el log de sortida de l'execució de cada script escrit en llenguatge *Python*, en un fitxer de text amb extensió *.log*.

Script per fer les peticions als recursos CoAP i pujar les dades obtingudes al full de càlcul de Google Drive:

```
# -*- coding: utf-8 -*-
import os, time, sys
import json
import gspread            ## To connect to GoogleSpreadsheet and change the cells values.
from oauth2client.service_account import ServiceAccountCredentials            ## To
authenticate to Google Drive.
from datetime import datetime            ## To extract the timestamp of the system.

import logging                           ## To see the output logs in the execution time.

from twisted.internet.defer import Deferred        ## Twisted Framework -> Event-driven
```

```
network programming framework.
from twisted.internet.protocol import DatagramProtocol
from twisted.internet import reactor
from twisted.python import log

import txthings.coap as coap                ## CoAP library for Twisted framework.
import txthings.resource as resource

from ipaddress import ip_address            ## To use the ipaddress module to specify the CoAP
Server IPv6 address.

logging.basicConfig(level=logging.INFO)


class Agent():

  recurs=0                          ## To save the resource ID.
  coapserveripv6addr=0              ## To save the IPv6 Address of the CoAP Server.

  def __init__(self, protocol):
      self.protocol = protocol
                  self.coapserveripv6addr = addr.read(23)    ## Read the IPv6 Address of the
CoAP Server from the file specified in the open("XXX.txt") function.
      reactor.callLater(1, self.requestResource5TE)        ## Call the request function of the 5TE
sensor resource after 1s.

  def requestResource5TE(self):
      request = coap.Message(code=coap.GET, mtype=coap.CON)        ## Specify the request
options.
      request.opt.uri_path = ('sensors', 'D5TE')     ## Resource URI.
      self.recurs = str(request.opt.uri_path[1])      ## Save the resource ID.
      request.remote = (ip_address(self.coapserveripv6addr), coap.COAP_PORT)          ##
Specify the IPv6 addr. of the CoAP Server and the CoAP port.
      d = protocol.request(request)                    ## Make the request.
      d.addCallback(self.printResponse)                ## Call the printResponse() callback function.
      d.addErrback(self.noResponse)                    ## Call if we don't have a response.
      reactor.callLater(10, self.requestResource5TM1)    ## Call the request function of the 5TM
1 sensor resource after 10s.

  def requestResource5TM1(self):
      request = coap.Message(code=coap.GET, mtype=coap.CON)
      request.opt.uri_path = ('sensors', 'D5TM1')
      self.recurs = str(request.opt.uri_path[1])
      request.remote = (ip_address(self.coapserveripv6addr), coap.COAP_PORT)
      d = protocol.request(request)
      d.addCallback(self.printResponse)
      d.addErrback(self.noResponse)
      reactor.callLater(10, self.requestResource5TM2)

  def requestResource5TM2(self):
      request = coap.Message(code=coap.GET, mtype=coap.CON)
      request.opt.uri_path = ('sensors', 'D5TM2')
      self.recurs = str(request.opt.uri_path[1])
      request.remote = (ip_address(self.coapserveripv6addr), coap.COAP_PORT)
      d = protocol.request(request)
      d.addCallback(self.printResponse)
      d.addErrback(self.noResponse)
      reactor.callLater(10, self.requestResourcePT100)

  def requestResourcePT100(self):
```

```
        request = coap.Message(code=coap.GET, mtype=coap.CON)
        request.opt.uri_path = ('sensors', 'PT100')
        self.recurs = str(request.opt.uri_path[1])
        request.remote = (ip_address(self.coapserveripv6addr), coap.COAP_PORT)
        d = protocol.request(request)
        d.addCallback(self.printResponse)
        d.addErrback(self.noResponse)
        reactor.callLater(10, self.requestResourceBATT)

    def requestResourceBATT(self):
        request = coap.Message(code=coap.GET, mtype=coap.CON)
        request.opt.uri_path = ('sensors', 'Battery')
        self.recurs = str(request.opt.uri_path[1])
        request.remote = (ip_address(self.coapserveripv6addr), coap.COAP_PORT)
        d = protocol.request(request)
        d.addCallback(self.printResponse)
        d.addErrback(self.noResponse)
        reactor.callLater(10, self.requestResourceLQI)

    def requestResourceLQI(self):
        request = coap.Message(code=coap.GET, mtype=coap.CON)
        request.opt.uri_path = ('sensors', 'Radio')
        self.recurs = str(request.opt.uri_path[1])
        request.remote = (ip_address(self.coapserveripv6addr), coap.COAP_PORT)
        d = protocol.request(request)
        d.addCallback(self.printResponse)
        d.addErrback(self.noResponse)

    def printResponse(self, response):
        timestamp = time.strftime("[%Y-%m-%d %H:%M:%S]", time.localtime(time.time()))  ##
Extract the timestamp of the system.
        print response.remote       ## Print the remote server info (IPv6 address and used PORT)
        print timestamp + '::' + response.payload

        # Save the reading obtained from the sensor in the msg variable.
        nodeid = str(response.remote[0])   ## Save the IPv6 addr. of the CoAP Server.
        msg = str(response.payload)        ## Save the message content (BATT, LQI, 5TE, 5TM
1, 5TM 2, or PT100).
        msgStr = str(msg)                  ## Save the message to edit.

        if (self.recurs == 'D5TE'):        ## If the required resource is D5TE.
            nummsg, temp, ec, vwc = msgStr.split("::")     ## Separate the received parameters.

            data = open("d5te.txt", "a")     ## Local file to save the readings "d5te.txt". Option 'a'
is to append a new line to the file.
            data.write(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + temp + ' ' + ec + ' ' + vwc +
'\n')                ## Write a new line to the file with the previous obtained values.
            print(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + temp + ' ' + ec + ' ' + vwc + '\n')

            wks = sht.worksheet('D5TE')    ## Select the sheet to publish the data to the
GoogleSpreadsheet.
            wks.append_row([timestamp, nodeid, nummsg, temp, ec, vwc])          ## Append a
new row to the D5TE sheet with the values.

        elif (self.recurs == 'D5TM1'):
            nummsg, temp, vwc = msgStr.split("::")

            data = open("d5tm1.txt", "a")
            data.write(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + temp + ' ' + vwc + '\n')
            print(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + temp + ' ' + vwc + '\n')
```

```
            wks = sht.worksheet('D5TM1')
            wks.append_row([timestamp, nodeid, nummsg, temp, vwc])

        elif (self.recurs == 'D5TM2'):
            nummsg, temp, vwc = msgStr.split("::")

            data = open("d5tm2.txt", "a")
            data.write(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + temp + ' ' + vwc + '\n')
            print(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + temp + ' ' + vwc + '\n')

            wks = sht.worksheet('D5TM2')
            wks.append_row([timestamp, nodeid, nummsg, temp, vwc])

        elif (self.recurs == 'PT100'):
            nummsg, value = msgStr.split("::")

            data = open("pt100.txt", "a")
            data.write(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + value + '\n')
            print(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + value + '\n')

            wks = sht.worksheet('PT100')
            wks.append_row([timestamp, nodeid, nummsg, value])

        elif (self.recurs == 'Battery'):
            nummsg, value = msgStr.split("::")
            data = open("batt.txt", "a")
            data.write(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + value + '\n')
            print(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + value + '\n')

            wks = sht.worksheet('BATT')
            wks.append_row([timestamp, nodeid, nummsg, value])

        elif (self.recurs == 'Radio'):
            nummsg, value = msgStr.split("::")
            data = open("radio.txt", "a")
            data.write(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + value + '\n')
            print(timestamp + ' ' + nodeid + ' ' + nummsg + ' ' + value + '\n')

            wks = sht.worksheet('RADIO_LQI')
            wks.append_row([timestamp, nodeid, nummsg, value])
            reactor.stop()          ## Stop the reactor and close the connection to the CoAP
Server.

        else:
            print 'The resource does not exist.'
            reactor.stop()


        data.close()        ## Close the file where we save the readings in local.
        addr.close()        ## Close the file that contains the IPv6 addr. of the CoAP Server.

def noResponse(self, failure):
        print 'Failed to fetch the resource'
        #failure
        reactor.stop()


scope = ['https://spreadsheets.google.com/feeds']
```

```
## Specify the directory and the filename of the .json file.
credentials = ServiceAccountCredentials.from_json_keyfile_name('/home/pi/GoogleDocs/tfg-
hivernacle-1d2f84f6ac6f.json', scope)
gc = gspread.authorize(credentials)

sht = gc.open('test')       ## Open the sheet 'test' saved in Google Drive (create if we don't have
it).

## Store data locally in the BR.
os.chdir("/home/pi/GoogleDocs/data")

addr = open("/home/pi/GoogleDocs/coapserver_ipv6_addr.txt")  ## Open the
'coapserver_ipv6_addr.txt' file to extract the IPv6 addr. of the CoAP Server.

log.startLogging(sys.stdout)       ## To see the output log of the script in the execution time.
endpoint = resource.Endpoint(None)
protocol = coap.Coap(endpoint)
client = Agent(protocol)


reactor.listenUDP (0, protocol, interface = '::0')     ## Listen the UDP protocol.
reactor.run()                                          ## Open the socket connection.
```

Fig. XII.II - Arxiu *data2googledocs.py*

Per automatitzar les peticions als recursos CoAP i la pujada de les dades obtingudes al full de càlcul de Google Drive s'ha d'executar l'script de la figura XII.II mitjançant el dimoni *cron*.

ANNEX XIII – FUNCIONAMENT DEL SISTEMA

En aquest annex es mostren els mètodes utilitzats per comprovar el funcionament dels diferents mòduls que formen el sistema que s'ha desplegat a l'hivernacle, així com el funcionament del sistema complet.

XIII.I Comprovació del funcionament del mòdul de sensors

Degut als diferents tipus de sensors utilitzats en aquest sistema (Decagon 5TE i 5TM i RTD PT100) s'han d'utilitzar mètodes de comprovació diferents, ja que els sensors Decagon són digitals, mentre que el sensor PT100 és analògic.

XIII.I.I Sensors Decagon

Per comprovar el funcionament dels sensors Decagon hi ha dos mètodes possibles, el primer utilitzant la llibreria *libmc1322x* (veure referència bibliogràfica [12]) i la informació presentada als annexos II i III d'aquesta memòria. El segon utilitzant un client CoAP que realitzi les peticions als recursos CoAP del servidor allotjat al node-sensor. Aquest client està implementat mitjançant el complement Copper pel navegador Mozilla Firefox.

XIII.I.I.I Llibreria libmc1322x

El codi utilitzat per comprovar si es reben les dades dels sensors és el de la figura XIII.I.

```
#include <mc1322x.h>
#include <board.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "tests.h"
#include "config.h"

#define DELAY 8000000

#define PWR_ON 0x1000000
#define PWR_OFF 0xFEFFFFFF // ~on


//-------------------------------------------------------------------------------//
void init_config (void) {

   GPIO->PAD_PU_SEL.U2RX = 1; // Pull-ups/pull-downs enabled for GPIO_19/UART2_RX
   *GPIO_FUNC_SEL1 |= 0x30000;       // Select the function of GPIO_24/KBI_2
*GPIO_PAD_DIR0 |= 0x1000000;         // Setting as output the GPIO_24/KBI_2.

   uart_init(UART2, 1200); // To read the data from the sensor
```

```
  printf("GPIO configured and UART initiated. \n");
}
//---------------------------------------------------------------------------------//
void read_sensor (void) {

  uint8_t c = 0x00;                 // To save one character readed from UART2.
  uint8_t d = 0x00;                 // To view the discarded characters.
  uint16_t j = 0x00;                // Index for raw_data[] uint8_t array.
  uint8_t raw_data [20] = {0};      // Vector to store raw values readed from UART2.

  *GPIO_DATA0 |= PWR_ON; // Put '1' in the GPIO_24/KBI_2.

  static struct timer readtimer;    // Declare a timer to limit the maximum reading sensor time.

  timer_set(&readtimer, 3 * CLOCK_SECOND); // Set the time of the reading timer.

  printf("Waiting data. \n");

  while ( uart2_can_get() ) {       // To discard the previous bad chars and clean the UART2 SW
buffer to read the desired characters.

    d = uart2_getc();
  }
  while( (c != 0x0A) && (j < 20) ) {          // Read while the readed char is different of '0x0A'
special char, and the number of readed chars is lower than 20.

    if(uart2_can_get()) { // Read from UART2 if some information is received.

      c = uart2_getc();
      raw_data[j] = c;
      printf("Raw_data: %02X , j=%d \n", raw_data[j], j);
      j++;
    }
    if (timer_expired(&readtimer)) { break;}         // If read timer is expired quit from the while.
  }

  *GPIO_DATA0 &= PWR_OFF;           // Put '0' in the GPIO_24/KBI_2.
  timer_reset(&readtimer);          // Reset the readtimer.
}
//---------------------------------------------------------------------------------//
void main(void) {

        uint32_t i = 0;    //Timer index
        printf("Initial setup. \n");
        init_config();
        printf("Initial setup done. \n");
        uart_init(UART1, 115200);         // Communication between the Econotag running
program and the console

        while(1) {

          printf("Inside the while. \n");

          read_sensor();

          for (i=0; i < DELAY; i++) {continue; }  // timer
        }
}
```

Fig. XIII.I – Codi per llegir dades dels sensors Decagon 5TE i 5TM.

El codi de la figura XIII.I realitza les tasques següents:

1. Configura els registres del GPIO i ports UART que s'utilitzen.

2. Alimenta el sensor.

3. Inicia el comptador *readtimer* de 3 s de duració.

4. Descarta caràcters previs dolents i neteja el buffer emprat pel port UART2 per llegir els caràcters que envia el sensor.

5. Llegeix els caràcters que envia el sensor mentre el caràcter llegit no sigui el caràcter especial *'0x0A'* i no s'hagin llegit més de 20 caràcters. També imprimeix els caràcters llegits per pantalla i els emmagatzema a l'array *'raw_data[]'*.

6. Reseteja el comptador *readtimer*.

XIII.I.I.II Client CoAP

Si utilitzem el complement Copper per comprovar el funcionament dels sensors Decagon és necessari seguir els passos de l'apartat 4.4 d'aquesta memòria. Mitjançant la pàgina web *sensors.html* del SW 6LBR (veure figura 4.15), que anuncia el BR, accedim als recursos disponibles del servidor CoAP (veure figura 4.16). Des de la pàgina on estan visibles els recursos del servidor realitzem peticions als recursos CoAP per veure si es llegeixen les dades dels sensors i s'envien correctament

A la figura XIII.II es pot veure com el servidor CoAP envia els paràmetres que ha llegit el sensor 5TE. Si la lectura del sensor és errònia, o el sensor no està connectat, els valors que retornarà el servidor seran tots els paràmetres amb el valor 9999.

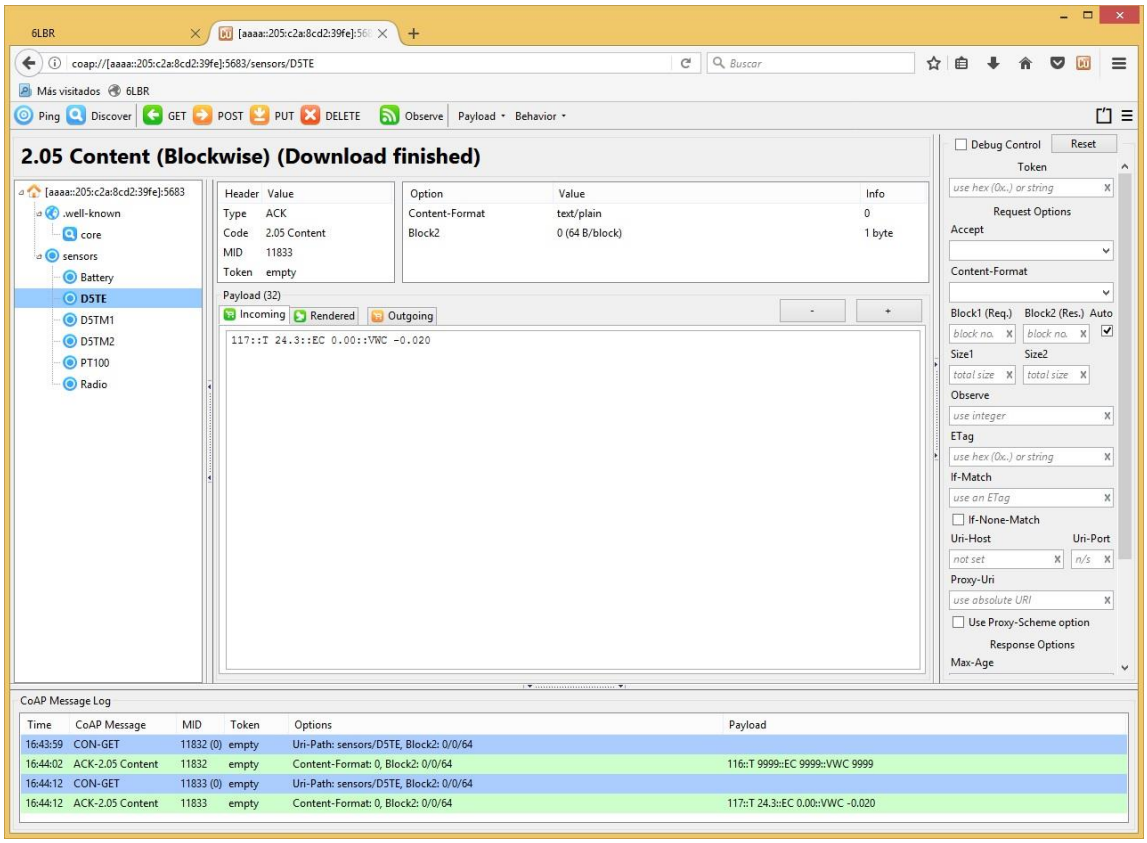

Fig. XIII.II Client CoAP. Petició al recurs CoAP '*D5TE*'.

XIII.I.II Sensor RTD PT100

Per comprovar el funcionament del sensor RTD PT100 hi ha dos mètodes possibles, el primer utilitzant la llibreria *libmc1322x* (veure referència bibliogràfica [12]) i la informació presentada a l'annex IV d'aquesta memòria. El segon utilitzant un client CoAP que realitzi les peticions al recurs CoAP del servidor allotjat al node-sensor. Aquest segon mètode segueix els passos descrits a l'apartat XIII.I.I.II d'aquest annex.

XIII.II Funcionament del BR

El primer mètode de comprovació del funcionament del BR es pot revisar a l'apartat 4.4 d'aquesta memòria, pel que passem a veure el segon mètode per comprovar que el BR està funcionant.

El segon mètode per comprovar que el BR està funcionant correctament consisteix en realitzar un ping a l'adreça IPv6 que anuncia el BR, per defecte *bbbb::100*, des del PC on ens connectem via Ethernet al BR. Per fer-ho, necessitem configurar l'ínterfície Ethernet del PC amb els paràmetres que apareixen a la figura 4.13 d'aquesta memòria.

Un cop configurada l'interfície Ethernet del PC obrim una finestra de comandes des de Windows i escrivim la comanda següent:

ping bbbb::100

Si obtenim resposta és què el BR està funcionant correctament, pel contrari, haurem de revisar que el cable Ethernet estigui ben connectat i que el servei *6lbr* estigui corrent a la RPi.

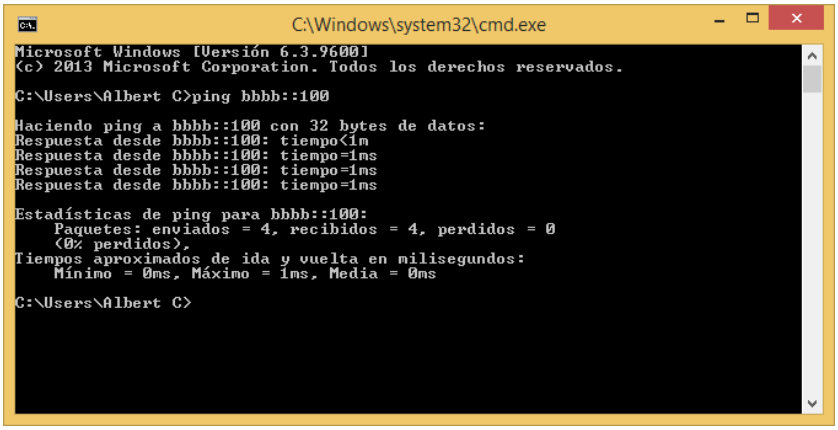

Fig. XIII.III Comanda *ping* executada del PC al BR.

XIII.III Funcionament del servidor CoAP

A l'apartat 4.4 d'aquesta memòria es pot revisar el primer mètode de comprovació per revisar que el servidor CoAP està en funcionament. Aquest mètode permet revisar si el BR detecta el servidor CoAP mitjançant el complement Copper pel navegador Mozilla Firefox.

Un altre mètode per comprovar si el servidor CoAP està en funcionament és mitjançant la comanda *ping6*. Si el BR té una ruta IPv6 definida cap a la WSN podem fer un *ping* a l'adreça IPv6 del servidor CoAP per veure si ens contesta (veure figura XIII.IV), el què ens indicarà que està en funcionament.

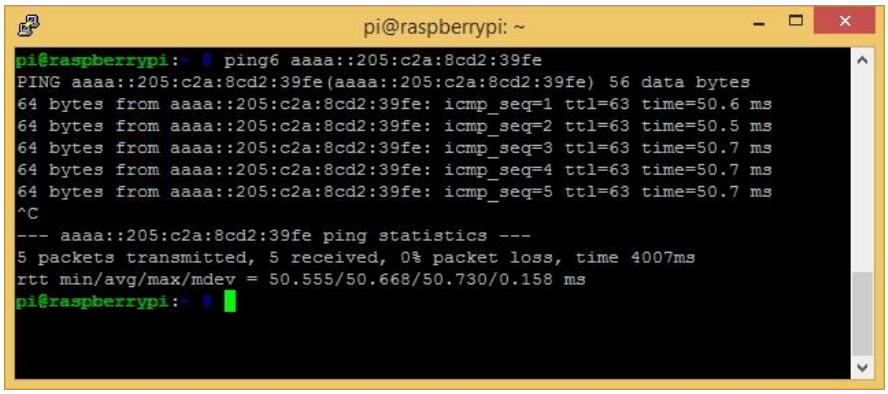

Fig. XIII.IV Comanda *ping6* executada del BR al servidor CoAP.

Després de veure els mètodes per comprovar els diferents mòduls que formen el sistema, passem a veure el funcionament del sistema complet.

XIII.IV Funcionament del sistema complet

Per comprovar que el sistema complet funciona correctament podem revisar el full de càlcul de Google Drive per veure si es pugen les dades correctament, i si aquestes dades tenen uns valors dins dels marges esperats.

A les figures XIII.V i XIII.VI es pot revisar els gràfics obtinguts de les dades llegides pel sensor Decagon 5TE, i de l'indicador *LQI*, durant 14 dies consecutius.

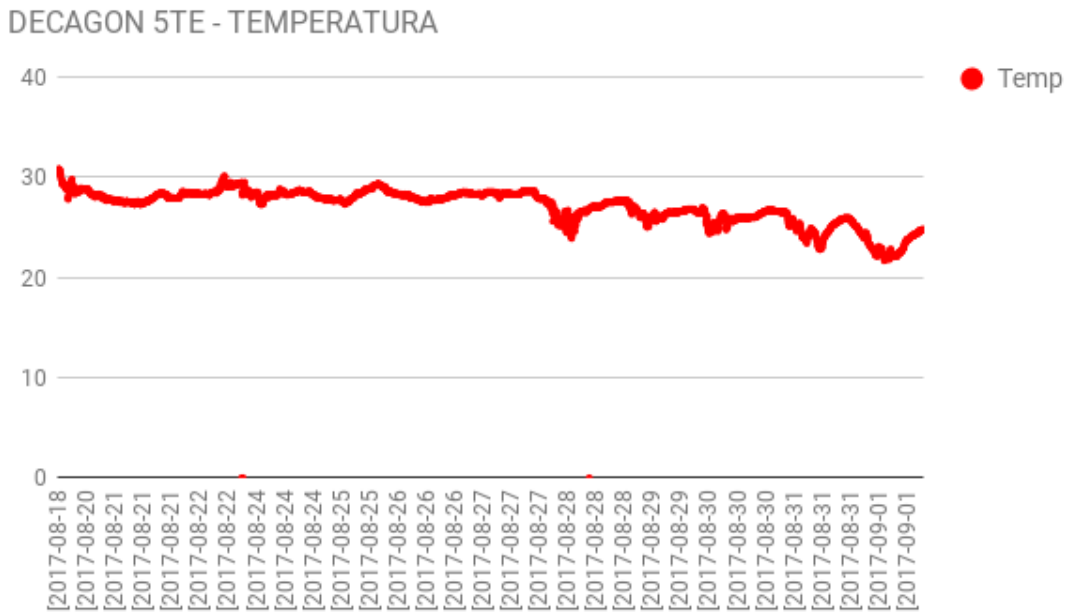

Fig. XIII.V Gràfic de temperatura obtingut del sensor Decagon 5TE.

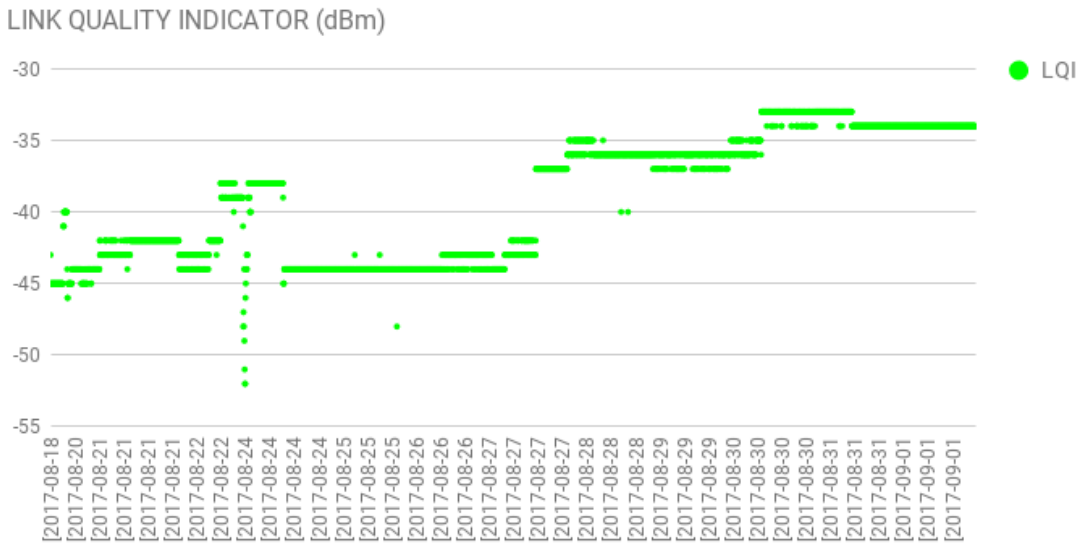

Fig. XIII.VI Gràfic de l'indicador *LQI*